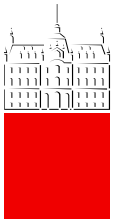
Univerza v Ljubljani
Fakulteta za računalništvo in informatiko



MARKO POŽENEL

Inteligentno razpletanje sej pri izgradnji spletnega podatkovnega skladišča

DOKTORSKA DISERTACIJA

Mentor: prof. dr. Viljan Mahnič
Somentor: doc. dr. Matjaž Kukar

Ljubljana, 2010

Povzetek

Uporaba svetovnega spleta se je v zadnjih letih močno povečala. Povečuje se število spletnih mest, pojavljajo se nove aplikacije in načini izrabe spleta. Podjetja potrebujejo splet za poslovanje in promocijo svojih izdelkov, posamezniki preko spleta učinkovito dostopajo do različnih storitev. V boju s tekmeci je za podjetja pomembno, da čim bolje izkoristijo možnosti, ki jih nudi splet. V novih okoliščinah uporabniki lahko enostavno primerjajo ponudbo in se odločajo za najbolj ugodne izdelke in storitve. Težje je torej ohraniti obstoječe stranke in privabiti nove. V takih okoliščinah bodo uspešna predvsem tista spletna mesta, ki vedo, kdo so njihove stranke in kakšne so njihove potrebe.

Analiziranje obnašanja strank je postalo eno izmed pomembnih analiz podatkov. Analize obnašanja zajemajo preprostejše statistike, kot je pogostost dostopa do neke strani, in bolj sofisticirane oblike, kot je najbolj običajen sprehod uporabnika po spletnem mestu. Glavni vir podatkov za analize obnašanja predstavljajo podatki o klikotoku, ki jih najdemo v dnevnikih spletnih strežnikov. Za analizo teh podatkov pogosto gradimo spletna podatkovna skladišča, pri tem pa naletimo na problem, da so podatki o klikotoku slabo strukturirani in velikokrat pomanjkljivi. Zato jih moramo – preden z njimi napolnimo podatkovno skladišče – najprej ustrezno obdelati in preoblikovati. Od kakovosti podatkov v podatkovnem skladišču je namreč odvisna kakovost vzorcev obnašanja uporabnikov, ki jih odkrijemo med analizo.

Postopek predprocesiranja klikotoka je velikokrat osnovan na hevrističnih predpostavkah o uporabi strani in je zato podvržen napakam. Doslej je bilo predlaganih veliko metod za predprocesiranje, vendar zanesljiva rekonstrukcija sej še vedno ostaja izziv. Z razmahom brskalnikov nove generacije, ki ponujajo brskanje v zavihkih, se je povečalo število prepletenih sej. Prepleteno sejo ustvari uporabnik, ki istočasno brska po spletni strani z več odprtimi okni brskalnika. V prepleteni seji časovno zaporedje klikov uporabnika ne sovpada več z zaporedjem obiskanih strani v uporabniški seji. Prepletene seje kvarno vplivajo na rezultate analiz, zato jih je

potrebno razplesti. Običajno prepletene seje generirajo za nas pomembni uporabniki. V doktorski disertaciji se ukvarjamo s problemom razpletanja prepletenih sej. Cilj doktorske disertacije je podrobno preučiti prepletene seje in razviti učinkovite metode za njihov razplet, da bi tako zagotovili kakovostne podatke za polnjenje podatkovnega skladišča. V ta namen najprej osvetlimo problem razpletanja sej s teoretičnega vidika, opišemo značilnosti prepletenih sej in definiramo teoretično ozadje. Na podlagi kombinatoričnih rezultatov pokažemo zapletenost problema. V osrednjem delu disertacije nato predstavimo dve metodi za razpletanje: metodo z uporabo markovskega modela prvega reda in metodo z uporabo algoritma RFBS. Ta metoda daje kot rezultat razplet z največjo verjetnostjo zaporedij strani znotraj razpletenih sej.

Obe metodi smo ovrednotili tako na eksperimentalnih, umetno generiranih podatkih, kot na realnih podatkih iz spletnega študijskega informacijskega sistema e-Študent in spletne trgovine EnaA. Za ovrednotenje smo uporabili več različnih kriterijev, ki temeljijo na podlagi merjenja podobnosti dveh zaporedij simbolov. Rezultati kažejo, da oba postopka dobro razpletata seje, boljše rezultate pa pričakovano daje postopek, ki uporablja preiskovanje prostora stanj. Razvita postopka razpletanja sta splošna, tako da ju lahko uporabimo na poljubnem viru podatkov o klikotoku.

Ključne besede: seja HTTP, kakovost podatkov, klikotok, markovski model, stohastični model, obnašanje uporabnikov, osejevanje, prepletena seja, RBFS, razpletanje sej, podatkovno skladišče

Abstract

In the past decades, World Wide Web (WWW) has become one of the main sources of information. The number of web sites and web pages is increasing, new applications and ways of web usage emerge. Companies need web sites to reach customers and sell their products, institutions furnish information about their services, individuals can effectively access various services over the Internet. However, companies with web presence need to make better use of opportunities offered by the web. Under the new circumstances customers can easily compare offers and choose favourable products and services. It is therefore difficult to attract new customers and retain the existing ones. It is evident that only those web sites that know their customers and understand the needs of their customers will prevail.

Analysing users' behavior has become an important part of web page data analysis. Users' behavior analysis includes some more or less straightforward statistics, such as page access report, and some more sophisticated forms of analysis, such as finding the most common path through a web site. The main source of data for user behavior analysis is clickstream data that can be obtained from web server log files. For analysing clickstream data we often build data webhouses, where we encounter the problem of clickstream data quality. Clickstream data are often inadequately structured and show an incomplete picture of users' activity. For this reason, several preprocessing tasks have to be performed prior to loading the clickstream data to a data webhouse. The quality of the patterns discovered in data behavior analysis largely depends on the quality of the data in the data webhouse.

The clickstream preprocessing process is often based on heuristic rules and assumptions about the site's usage and is therefore prone to errors. Many methods for clickstream preprocessing have been proposed, but reliable session reconstruction still remains a challenge. With the advent of new browser generations that offer tabbed browsing feature, the number of interleaved sessions has increased. An interleaved session is generated by a user who is concurrently browsing a web site in two or more web sessions (browser windows). In an interleaved session, temporal

order of user clicks does not necessarily correspond to the sequence of browsed pages in a web session. Interleaved sessions have a negative effect on data analysis quality, so they must be separated. Interleaved sessions tend to be more often created by users important to us.

In the thesis we deal with the problem of interleaved HTTP sessions. The goal of the thesis is to examine interleaved sessions and develop efficient methods for their separation so that we could provide quality data for data webhouse loading. To this end, we first consider the problem of separating interleaved sessions from a theoretical point of view. Then we describe characteristics of interleaved sessions and define theoretical background. In the central part of the thesis, we present two new separation methods: a method based on trained first-order Markov model and a method that uses recursive best-first heuristic search. The second method gives as a result the session separation with the highest probability of page sequences for the separated sessions.

We evaluated both developed methods on experimental, artificially generated data and on two real-world clickstream data sources: the university student records information system e-Študent and the web shop EnaA. For evaluation we used several different criteria, which are based on symbol sequence similarity. The results clearly show that both methods adequately separate interleaved sessions. The method that uses best-first search gives better results. Both developed methods are general, so they can be used on any clickstream data source.

Keywords: HTTP session, data quality, clickstream, Markov model, stochastic model, user behavior, sessionization, RBFS, interleaved session, session separation process, data warehouse

I Z J A V A O A V T O R S T V U

doktorske disertacije

Spodaj podpisani *Marko Poženel*
z vpisno številko *24950411*

sem avtor doktorske disertacije z naslovom

Inteligentno razpletanje sej pri izgradnji spletnega podatkovnega skladišča

S svojim podpisom zagotavljam, da:

- sem doktorsko disertacijo izdelal samostojno pod vodstvom mentorja *prof. dr. Viljana Mahniča*

  in somentorstvom *doc. dr. Matjaža Kukarja*

- so elektronska oblika doktorske disertacije, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko doktorske disertacije

- in soglašam z javno objavo elektronske oblike doktorske disertacije v zbirki "Dela FRI".

V Ljubljani, dne 12.8.2010 Podpis avtorja:

Zahvala

Iskreno se zahvaljujem mentorju prof. dr. Viljanu Mahniču za vodenje in pomoč pri vseh težavah, ki so se pojavile v času nastajanja tega dela. Prof. Mahnič me je ves čas stoodstotno podpiral in mi omogočal ustrezno ustvarjalno okolje. Iskreno se zahvaljujem tudi somentorju doc. dr. Matjažu Kukarju za ves trud in čas ter številne ideje in usmeritve pri razvoju postopkov. Doc. Kukar mi je pomagal pri izboljšanju pristopa k raziskovalnem delu. Brez njune pomoči tega dela ne bi bilo. Hvala tudi ostalima članoma komisije, prof. dr. Borutu Robiču in dr. Matjažu Gamsu za trud, ki sta ga vložila v pregled doktorske disertacije.

Hvala g. Aljoši Domijanu iz podjetja Gambit Trade d.o.o., ki mi je omogočil preizkus razvitih postopkov na problemski domeni spletne trgovine in g. Saši Bradaču za vso tehnično pomoč in posredovane informacije.

Zahvaljujem se tudi sodelavcema v LTPO, Luki Fűrstu in dr. Igorju Rožancu, ki sta omogočila ustvarjalno vzdušje in me moralno podpirala v času nastajanja disertacije. Luki Fűrstu se zahvaljujem za obsežne diskusije o raznih problemskih domenah, dr. Igorju Rožancu pa za fleksibilnost pri študijskem procesu in marsikatero vzpodbudno besedo, ki je bila zlata vredna.

Zahvala gre tudi staršem in starim staršem za vso podporo v času študija. Zahvaljujem se tudi vsem, ki jih v zahvali pomotoma nisem navedel, so pa kakorkoli pripomogli k nastanku tega dela.

Kazalo

Slike

Tabele

POGLAVJE 1

Uvod

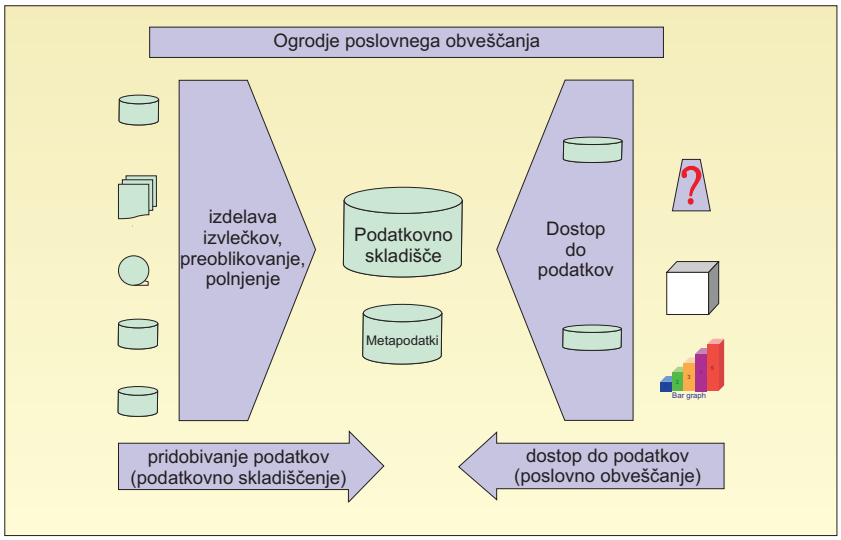
Skrb za kakovost podatkov in zavedanje njihovega pomena pri poslovnih odločitvah je prisotna že dlje časa. Relativno nov pristop je uporaba istih podatkov za več različnih namenov, ki se zelo razlikujejo od njihovega prvotnega namena uporabe. V ta namen se pogosto uporabljajo podatkovna skladišča . Podatkovna skladišča hranijo velike količine podatkov, ki jih potrebujemo pri poslovnih odločitvah. Podatkovno skladišče je odvisno od zagotavljanja kakovosti podatkov v vseh fazah pod.skladiščenja: - načrtovanja, implementacije in vzdrževanja [6]. Nič bolj ne pripomore k neuspehu podatkovnega skladišča kot nepravilno razumevanje zahtev in zapostavljanje skrbi za kakovost vhodnih podatkov.

Kakovost podatkov je glavna skrb tako transakcijskih sistemov kot podatkovnih skladišč [60, 32]. Zagotavljanje točnosti, pravočasnosti, konsistentnosti in celovitosti podatkov je lahko težavno tudi znotraj podjetij, kjer z njimi upravljajo izurjeni uporabniki. Ti problemi se povečajo, če imamo opravka z informacijskimi sistemi (IS), ki so na voljo strankam, kupcem in ostalim dobaviteljem. Rezultat je za podatkovno skladišče, ki ima za vir takšne IS, lahko poguben. Istočasno je zavračanje nekakovostnih podatkov iz transakcijskih sistemov zaradi težav s sinhronizacijo neustrezno, še posebej, če se podatkovno skladišče napaja iz več raznovrstnih viriv. Metode za čiščenje in preoblikovanje podatkov so se v takšnih primerih izkazale za uspešne, vendar je potrebno posvetiti še več pozornosti kakovosti podatkov [62].

V doktorski disertaciji smo se osredotočili na določene postopke za zagotavljanje kakovosti vhodnih podatkov v fazi načrtovanja in implementacije spletnega podatkovnega skladišča.

1.1 Pomen podatkovnih skladišč in poslovnega obveščanja v sodobnih informacijskih sistemih

Poslovno obveščanje (ang. Business Inteligence) je trenutno ena izmed prioritet direktorjev informatike (ang. CIO) [101]. Gartnerjeva raziskava med 1400 direktorji informatike je pokazala, da so bili projekti v povezavi s poslovnim obveščanjem v letu 2007 na njihovi prioritetni listi nalog. Viaene [96] navaja, da je bilo poslovno obveščanje v letu 2008 že tretje leto ena izmed glavnih tehnoloških prioritet CIO. Poslovno obveščanje je postalo ključno orodje pri učinkovitem poslovanju in inovacijah. Slika 1.1 prikazuje ogrodje poslovnega obveščanja. Vidimo, da je poslovno obveščanje proces, ki vključuje dva glavna procesa: *vnos podatkov v sistem* in *dostop do podatkov*.



Slika 1.1: Ogrodje poslovnega obveščanja. Vsebuje dve glavni aktivnosti, pridobivanje podatkov in dostop do podatkov. Povzeto po [101].

Podatkovno skladiščenje

Proces vnosa podatkov v sistem, ki ga imenujemo podatkovno skladiščenje , vključuje prenos podatkov iz heterogenih virov v integrirano podatkovno skladišče. Viri podatkovnega skladišča se običajno nahajajo na različnih platformah in v različnih podatkovnih strukturah. Viri podatkov se lahko torej nahajajo znotraj organizacije, lahko jih zagotavlja zunanji ponudnik ali pa so priskrbljeni s strani poslovnih partnerjev.

Podatke, pridobljene iz virov, moramo preoblikovati v obliko, primerno za uporabo v procesu odločanja. Proces prenosa podatkov v podatkovno skladišče je eden

največjih izzivov sistema poslovnega obveščanja. Proces zahteva 80% časa in napora in ustvarja 50% vseh nepredvidenih stroškov na projektu. Izziv izvira iz različnih vzrokov, kot so: slaba kakovost podatkov, ki jih pridobimo iz virov, dostopnih politik in slabe strukture podatkov v starejših sistemih (ang. legacy technology).

Podatki se pred zapisom v podatkovno skladišče ustrezno preoblikujejo. Konceptualno gledano je ideja podatkovnega skladišča zelo preprosta. Inmon in Hackatorn sta podatkovno skladišče definirala kot podatkovno zbirko, za katero velja predmetna usmerjenost, povezanost, časovna variantnost in nespremenljivost [39]. Podatkovno skladišče je torej shramba podatkov (ang. repository) v katerem hranimo vse podatke pomebne vodstvenemu kadru in iz katerih lahko izluščimo informacije, pomembne pri upravljanju podjetja [100]. Čeprav je to poenostavljen in idealiziran pogled na podatkovna skladišča, nam omogoča razumevanje ključnih izzivov in smeri razvoja. Osvetli glavni namen podatkovnega skladišča: podpora procesu odločanja na vseh ravneh s pomočjo pridobitve, integracije, preoblikovanja in interpretacije podatkov iz zunanjih in notranjih virov [65]. To so osnovne značilnosti podatkov v podatkovnem skladišču, ki jih je določil Bill Inmon [39].

V odvisnosti od arhitekture lahko podatkovno skladišče oskrbuje s podatki tudi odvisna področna podatkovna skladišča (ang. data marts), ki imajo bolj ozek pogled kot krovno podatkovno skladišče. Področna podatkovna skladišča se osredotočajo na posamezno funkcijsko področje, geografsko področje, aplikacijo ali organizacijsko enoto. Vzdrževanje podatkov, namenjenih procesu odločanja v podatkovnem skladišču ali področnih podatkovnih skladiščih, zagotavlja *eno samo različico resnice* – enoten pogled na podatke.

Pomembno je razlikovati med pojmom podatkovno skladišče kot shrambo integriranih podatkov (ang. data warehouse) od pojma podatkovno skladiščenje kot proces upravljanja in vzdrževanja podatkovnega skladišča (ang. data warehousing), ki določa metode in postopke kako se podatki zbirajo, preoblikujejo, združujejo in uporabljajo. Za proces posredovanja analitičnih podatkov uporabnikom pa se je v zadnjem času uveljavil izraz *poslovno obveščanje* (ang. Business Inteligence). Podatkovno skladišče je torej shramba obveščanja iz katerega lahko izpeljemo poslovno obveščanje [62].

Poslovno obveščanje

Podatki v podatkovni bazi sami kot taki nudijo podjetju le omejeno vrednost. Šele ko uporabniki in aplikacije lahko dostopajo do teh podatkov in sprejemajo poslovne odločitve, lahko podjetje v popolnosti izkoristi vrednost podatkovnega skladišča. Proces dostopa do podatkov pogosto označujemo z oznako BI (ang. Business Intelligence) ali poslovno obveščanje. Zelo pomembno je razlikovati med pojmoma obveščanje (ang. intelligence) in poslovno obveščanje (ang. business intelligence). Pojem *obveščanje* v širšem informacijskem smislu uporabljamo za informacije, pridobljene za pomoč pri izvajanju poslovnega procesa. S pojemom *poslovno obveščanje* pa se sklicujemo na sklepe in znanje pridobljeno iz pridobljenih informacij s pomočjo analiz na podlagi algoritmov. Poslovno obveščanje je sestavljeno iz apli-

kacij in uporabnikov, ki dostopajo do podatkovnega skladišča za izdelavo poročil, odkrivanja zakonitosti v podatkih in drugih analiz. Izraz poslovno obveščanje je leta 1958 predstavil pionir na področju informacijskih znanosti, Hans Peter Luhn [57]. Kljub zelo zgodnjim vizijam pa se poslovno obveščanje do nedavnega ni razvilo v glavno vejo informatike [4]. Večja pozornost je bila namenjena razvoju podatkovnih baz in podatkovnih skladišč.

1.1.1 Prednosti uporabe podatkovnih skladišč

Podatkovna skladišča tvorijo jedro sistemov poslovnega odločanja. Stroški, povezani s kadri, strojno in programsko opremo, niso zanemarljivi. Kljub temu pa prinašajo občutne konkurenčne prednosti podjetju. V začetku se zmanjšajo stroški infrastrukture IT (informacijske tehnologije) z odpravo nekaterih postopkov redundančnega izdelovanja izvlečkov podatkov. Odpravimo tudi potrebo po obstoju podvojenih neodvisnih področnih podatkovnih skladišč po podjetju. Na primer, podjetje 3M je upravičilo svoje večmiljonsko podatkovno skladišče na podlagi prihrankov pri usklajevanju področnih podatkovnih skladišč [102]. Primer podjetja, ki je v polni meri izkoristilo možnosti, ki jih ponuja podatkovno skladišče, je Harrah's. S pomočjo podatkovnega skladišča, ki hrani podatke o igrah na srečo, so preučili stranke, njihovo dobičkonosnost in priljubljenost posameznih iger. V skladu s temi podatki so izvajali promocijske aktivnosti in uporabljali v postopku odločanja. Podatkovno skladišče je bilo zelo uspešno. Investicija se je povrnila, podjetje je povečalo prihodke in svojo vrednost [101].

Podatkovna skladišča nudijo prihranek časa tudi dobaviteljem in strankam zaradi učinkovitejše dostave podatkov. Sam način uporabe podatkov v podatkovnem skladišču podjetja se spremeni, ko se podatkovno skladišče uporablja že daljši čas. Uporabnike najprej zanima, kaj se je zgodilo. V tem primeru podatkovno skladišče nudi neposredne in zelo merljive koristi uporabnikom. S časoma se začnejo uporabniki spraševati, zakaj se je to zgodilo. Na koncu pa se naučijo uporabljati podatkovno skladišče tudi za odkrivanje zakonitosti v podatkih in napovedovanje trendov. Prednosti in koristi podatkovnega skladišča postanejo bolj splošne, težje pa je neposredno ovrednotiti vse večjo množico koristi, ki jih prinaša.

Podatkovno skladišče spada na področje infrastrukture IT (osredotočeno na arhitekturo podatkov). Zagotavlja namreč osnovo za združevanje raznovrstnih množic iz notranjih in zunanjih virov podatkov, omogoča oddaljen dostop do podatkov, zagotavlja spoštovanje podatkovnih standardov, nudi odgovore na poslovna vprašanja in vzpodbuja strateško razmišljanje z omogočanjem odkrivanja zakonitosti v podatkih, aplikacij za upravljanje odnosov s strankami (ang. Customer Relationship Management – CRM) in drugih aplikacij. Uporabljajo se za iskanje odgovorov na poslovna vprašanja, ki zahtevajo analize in postopke, kot so vrtanje v globino, vrtilne tabele, razčlenjevanje in seštevanje podatkov, ki so najbolje podprti z OLAP (ang. On-Line Analytical Processing) orodji. Podatkovno skladišče omogoča učinkovito podporo odločanju in učinkovite rešitve poslovnega obveščanja, olajšuje uporabo

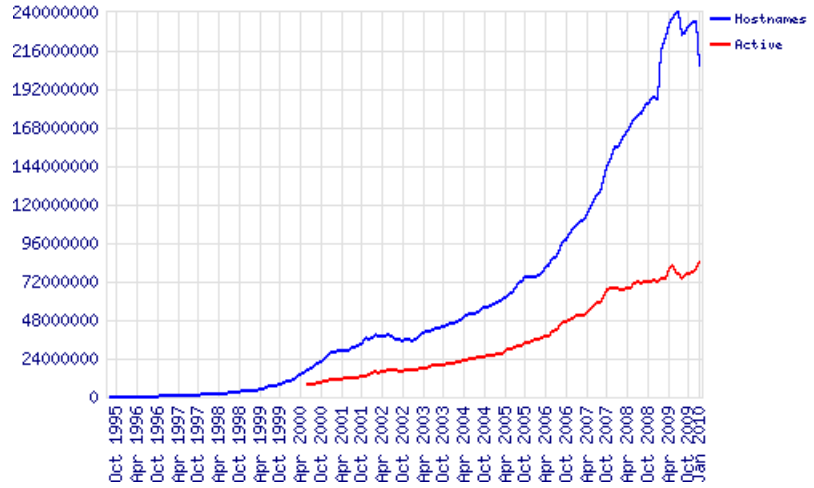
OLAP rešitev, zagotavlja integriteto, točnost, varnost in razpoložljivost podatkov, olajšuje uvedbo novih standardov, omogoča lažji dostop do podatkov in izboljšuje uporabniške storitve [35].

Inovacije morajo biti tiste, na katerih sloni konkurenčnost in preživetje v današnjem poslovnem okolju. Pravilna uporaba poslovnega obveščanja vodi do inovacij pri izdelkih, storitvah in procesih. Da bi lahko bile take inovacije uspešne, se morajo skladati s strategijo podjetja. Inovacije vodijo v organizacijske spremembe in v potrebe po upravljanju sprememb. Kritični sestavni del upravljanja s spremembami je razvoj podatkovnega skladišča za zajem podatkov o spremembah in podporo nenehnim ciklom izboljšav sistemov za podporo odločanju.

Kljub zelo zgodnjim raziskavam na področju ključnih faktorjev uspeha (ang. Critical Success Factors – CSF) in vodstvenih informacijskih sistemov (ang. Executive Information Systems – EIS) ter v zadnjem času aktualnim raziskavah na področju nadzornih plošč (ang. performance dashboards), povezava med podatkovnim skladiščenjem in strateškim odločanjem še vedno ostaja premalo raziskana [62].

1.2 Spletno podatkovno skladišče

Podatkovna skladišča so se pospešeno razvijala in postala zelo uspešna v devetdesetih letih prejšnjega stoletja. Razcvet spleta je zelo vplival tudi na podatkovna skladišča, ki so se morala prilagoditi novim zahtevam. Svetovni splet raste glede na število uporabnikov, intenzivnost uporabe in zapletenost zgradbe spletnih strani [66]. Slika 1.2 prikazuje porast števila spletnih mest v zadnjih petnajstih letih. Vsak



Slika 1.2: Prikaz števila medmrežnih domen med leti 1995 – 2010. Povzeto po [66].

dan nastane milijone zapisov o prehajanju uporbnikov med spletnimi stranmi in nastala je potreba po samodejnih postopkih ugotavljanja vzorcev v teh podatkih [47]. V novih okoliščinah je nastalo spletno podatkovno skladišče (ang. data webhouse).

Spletno podatkovno skladišče je podatkovno skladišče, ki hrani podatke o posameznih klikih uporabnikov na spletnih straneh – klikotoku (ang. cliskstream). Splet ponuja ogromen vir podatkov o uporabnikih, ki preko spletnih brskalnikov dostopajo do spletišč (ang. websites). Kljub temu, da je večina podatkov, povezanih s klikotokom uporabnikov, osnovnih in enostavnih, dajejo podroben vpogled v to, kako uporabniki prehajajo med spletnimi stranmi. Ta velik in nestruktuiran vir podatkov lahko prenesemo v spletno podatkovno skladišče, kjer ga struktuiramo in povežemo skupaj z ostalimi, bolj konvencionalnimi podatki, ki se že nahajajo v podatkovnem skladišču.

Sprememba, ki je ravno tako vplivala na podatkovno skladišče, ki hrani podatke o klikotoku, je način njegove uporabe. Poznavanje vedenja uporabnikov spletnih strani nam pomaga razumeti njihove želje in potrebe. Na podlagi njihovega preteklega obnašanja, shranjenega v spletnem podatkovnem skladišču v obliki klikotoka, lahko izvedemo prilagoditev spletnih vsebin uporabniku. Spletno podatkovno skladišče je torej postalo ključen del spletnega mesta. Hitrost dostopa do spletnega podatkovnega skladišča je torej postala ključna zahteva pri izgradnji. Nekatere operacije lahko terjajo več časa za izvedbo. Počasno odzivanje pa je nedopustno v okoljih, ki temeljijo na spletu. Pri spletnem podatkovnem skladišču je izrecno poudarjena zahteva po hitrem dostopu, izogibati se moramo vsakršnim prekinitvam.

Uporaba spletnih podatkovnih skladišč prinaša podjetjem, ki poslujejo preko spleta, številne prednosti. Predstavljajo enega izmed pomembnejših mehanizmov za identifikacijo ključnih strank in spoznavanje njihovih lastnosti. Združuje podatke iz različnih virov, nudi infrastrukturo za prepoznavanje obnašanja strank in njihovih prioritet. Omogoča analize več ključnih merljivih dejstev, kot so zahteve strank, učinkovitost oglaševanja, zbiranje demografskih podatkov in nakupovalne navade strank. Z njegovo pomočjo se lahko prilagodimo strankam in povečamo njihovo lojalnost. Spletno podatkovno skladišče predstavlja veliko prednost pred ostalimi rešitvami za analizo klikotoka zato, ker podatke o klikotoku povezuje z ostalimi podatki zunanjimi in notranjimi podatki podjetja. Nudi možnost poglobljenih analiz, kot je segmentacija uporabnikov ali nagnjenost uporabnikov k nakupu in je tudi dragocen vir povratnih informacij za izgradnjo kakovostnejših spletnih strani z vidika pridobivanja demografskih podatkov, izvajanja oglasnih akcij in učinkovitosti prodaje. Lopes in sod. [54] so spletno podatkovno skladišče uporabili za zaznavanje nenavadnih stanj in možnih zlorab sistema. Pokazali so, da je dosežen velik prihranek prostora, potreben za hranjenje podatkov o obnašanju uporabnikov. Izkazali so veliko analitsko fleksibilnost in prednosti v primerjavi z drugimi (ad hoc) načini spremljanja klikotoka.

Spletno podatkovno skladišče prinaša prednosti tako majhnim kot velikim, že uveljavljenim podjetjem. Začetna, najbolj otipljiva korist je krajši razvojni čas spletnega mesta in zmanjšanje stroškov ugotavljanja potreb uporabnikov. Na podlagi podatkov, ki jih zberemo in analiziramo, sproti prilagajamo strukturo spletnega mesta. Spletno podatkovno skladišče lahko apliciramo za vse spletne aktivnosti, še posebej za spletne trgovine. Kimball [43] navaja naslednje cilje spletnega podat-

kovnega skladišča:

- Hrani in nudi podatke o klikotoku in druge podatke o obnašanju, ki omogočajo razumevanje obnašanja strank.
- Je prilagojeno in povezano z drugimi področnimi podatkovnimi skladišči v podjetju v verigi dodane vrednosti in ga lahko uporabljamo skupaj z ostalimi deli podatkovnega skladišča podjetja.
- Predstavlja prožen in prilagodljiv vir informacij. S pojavitvijo novih virov podatkov in poslovnih vprašanj se ne poruši ravnovesje. Stare aplikacije lahko delujejo brez preprogramiranja, medtem ko se dodaja novo funkcionalnost.
- Podpira nove spletne medije kot so video, zvok in slika.
- Podatke podjetja daje na voljo strankam, zaposlenim in partnerjem obenem pa ščiti podatke podjetja pred nepooblaščeno uporabo.
- Tvori osnovo za spletno podprt proces odločanja.

1.3 Zagotavljanje kakovosti podatkov

Spletna podatkovna skladišča hranijo podatke o obnašanju uporabnikov spletnih strani. V mnogo podjetjih so postali ti podatki eden izmed najpomembnejših virov. Igrajo pomembno vlogo pri dnevnih, transakcijskih in pomembnih poslovnih odločitvah. Podatki o klikotoku predstavljajo glavni vir za analize o obnašanju uporabnikov [45]. Za pomembne poslovne odločitve potrebujemo zanesljive analize, kar zahteva primerne metode in kakovosten vir podatkov (spletno podatkovno skladišče). Kakovost vzorcev, odkritih v podatkih s pomočjo metod odkrivanja zakonitosti v podatkih, je namreč v največji meri odvisna prav od kakovosti vhodnih podatkov.

Spletno podatkovno skladišče polnimo s podatki o klikotoku. Največji izziv pri izgradnji spletnega podatkovnega skladišča je povezan s specifičnim virom podatkov in zagotovitvijo kakovostnih podatkov. Običajno najpogostejši vir podatkov o klikotoku predstavljajo dnevniške datoteke spletnih strežnikov [47]. Dnevniki spletnega strežnika so bili razviti za namen razhroščevanja spletnih strežnikov in ne kot vir podatkov za spletna podatkovna skladišča in nadalje za odkrivanje zakonitosti v podatkih in ostale analize. Podatki o klikotoku običajne spletne strani so nepopolni, vsebujejo veliko šuma, prehodi med posameznimi stranmi niso nazorno vidni. Pomanjkljivosti podatkov o klikotoku lahko v določeni meri odpravimo s kakovostnim postopkom obdelave pred polnjenjem v spletno podatkovno skladišče. To ni vedno možno, če so podatki preveč pomanjkljivi. Vir za spletno podatkovno skladišče so lahko podatki pridobljeni na:

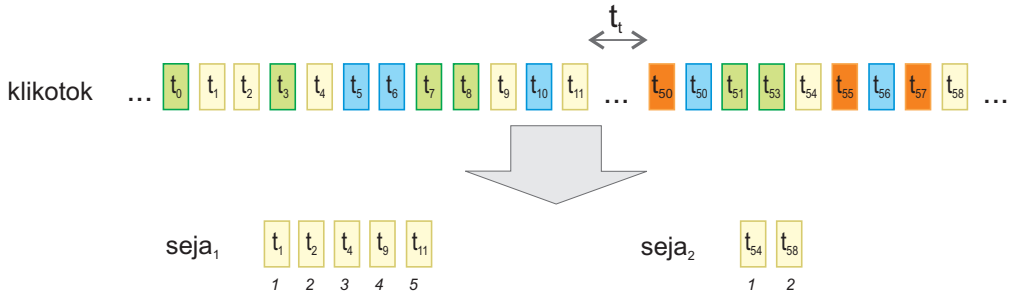
- uporabniških odjemalcih,

- spletnem strežniku (dnevniške datoteke) [71].

Podatki, pridobljeni na strežniku, imajo veliko prednost, da se podatki zbirajo za vse uporabnike. Odpade tudi potreba po instalaciji programske opreme na odjemalcih uporabnikov. Izbira se zdi sama po sebi logična, ker večina spletišč uporablja spletne strežnike s podporo generiranju dnevniških datotek. Dnevniške datoteke pa žal ne vsebujejo vseh podatkov o aktivnosti spletnih uporabnikov, ki jih potrebujemo. Ena večjih omejitev je odsotnost uporabniške seje, t.j. da uporabniške zahteve med seboj niso povezane. Omejitve, ki jih predstavljajo dnevniške datoteke spletnega strežnika so [47]:

- Spletni strežnik ne identificira posameznih uporabnikov in uporabniških sej.
- Podatke v dnevniški datoteki spletnega strežnika je potrebno povezati s transakcijskimi podatki.
- Zapisi v dnevniških datotekah ne vsebujejo podatkov o pomembnih dogodkih.
- Dnevniške datoteke spletnega strežnika ne hranijo podatkov o spletnih obrazcih (ang. web form).
- Zapisi v dnevniških datotekah vsebujejo naslove URL, ne pa semantične informacije o tem, kaj ti URL naslovi pomenijo.
- V dnevniških datotekah manjkajo podatki za dinamično generirane spletne strani.
- Vsebujejo veliko odvečnih in podvojenih podatkov.
- V dnevnikih spletnih strežnikov prav tako ne najdemo pomembnih informacij, ki jih sicer lahko pridobimo na druge načine. Na primer lokalni čas ali velikost zaslona na odjemalcu uporabnika.

Podatke o klikotoku moramo ustrezno zajeti, prečistiti in preoblikovati, preden jih zapišemo v spletno podatkovno skladišče. Ena izmed večjih težav pri tem je postopek osejevanja. V postopku osejevanja razpršene zapise o aktivnosti uporabnika v nekem časovnem obdobju združimo znotraj (uporabniške) seje. *Seja* ali obisk je definirana kot skupino akcij uporabnika od trenutka, ko je vstopil na spletno mesto do trenutka, ko je spletno mesto zapustil [88]. Omogoča nam pregled nad vsemi akcijami uporabnika na spletnem mestu v nekem časovnem obdobju. Seja predstavlja osnovni gradnik, ki se uporablja pri analizah obnašanja uporabnikov. Postopek osejevanja je prikazan na sliki 1.3. Akcije različnih uporabnikov so obarvane z različnimi barvami. Osredotočimo se na rumeno obarvane akcije. Predpostavimo, da je med akcijama t_{11} in t_{54} preteklo več časa, kot znaša časovna omejitev t_t. Sklepamo, da je uporabnik po zahtevi t_{11} zapustil spletno mesto in se ponovno vrnil v času t_{54}. Rumeno obarvane akcije se torej preoblikujejo v dve uporabniški seji.



Slika 1.3: Postopek osejevanja. Prikazan je klikotok s prepletajočimi se zahtevami uporabnikov in osejene rumeno obarvane zahteve.

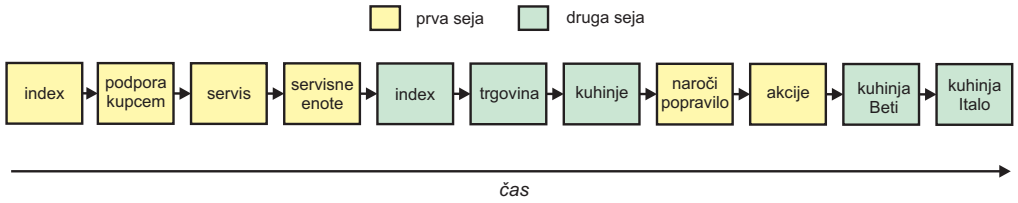
Slabo rekonstruirane uporabniške seje lahko hitro vodijo do napačno ugotovljenih vzorcev obnašanja uporabnikov in posledično vodijo v napačne sklepe. Kljub temu, da je bilo predlagano veliko metod za osejevanje [8, 107], verna rekonstrukcija uporabniških sej še vedno ostaja izziv. S pojavom brskalnikov, ki podpirajo brskanje v več oknih ali zavihkih, (ang. tabbed browsers) in povečevanjem števila naprednih uporabnikov, ki brskajo z več aktivnimi sejami na istem spletnem mestu, se je pojavil nov izziv – *prepletene seje*.

Prepletene seje

Vsaka zahteva uporabnika za določeno spletno stran se zabeleži v dnevniku spletnega strežnika. Zapisi v dnevniški datoteki so medseboj neodvisni in jih je potrebno med seboj povezati v postopku čiščenja in preoblikovanja. Posamezne zapise povezujemo v seje zato, da lahko vidimo celotno uporabnikovo sliko aktivnosti v določenem časovnem obdobju. Na podlagi individualnih zapisov v dnevniških datotekah ne moremo izvajati celovitejših analiz. Za to potrebujemo osejen klikotok. Šele iz uporabniške seje lahko namreč izločimo namen uporabnika in naloge, ki jih je hotel opraviti (npr. nakup, iskanje cenovno primerljivejšega izdelka v kategoriji, pregled novosti).

Uporabniki pogosto brskajo istočasno po določenem (predvsem obsežnejšem) spletnem mestu z več odprtimi okni (ali zavihki) spletnega brskalnika. V vsakem brskalnikovem oknu izvajajo akcije za dokončanje določene zaključene naloge. Običajno uporabniki tudi preklapljajo med posameznimi nalogami, tako da na vsaki delajo samo določen del časa. Kot primer navedimo uporabnika spletne aplikacije, ki v različnih uporabniških vlogah opravlja administrativne postopke. Za vsako uporabniško vlogo ima odprto ločeno okno brskalnika, ki jih izmenično uporablja. Tudi če bi predpostavili, da imamo na določenem spletnem mestu trenutno aktivnega samo enega takega uporabnika, imamo opravka s sočasnimi uporabniškimi sejami – za vsako brskalnikovo okno, po eno. V primeru, da spletni strežnik uporablja enega izmed standardnih oblik zapisovanja aktivnosti v dnevnik spletnega strežnika, bodo vse takšne sočasne seje v dnevniku spletnega strežnika vidne kot

ena sama dolga seja. Takšno sejo bomo poimenovali *prepletena seja*. Primer prepletene seje prikazuje slika 1.4. Uporabnik deska po spletni strani večjega ponudnika, ki ponuja izdelke in storitve s področja opreme prostorov. Prepletena seja je sestavljena iz dveh uporabniških sej. V prvi seji uporabnik preko spleta naroča popravilo pokvarjenega gospodinjskega aparata. Glede na obisk strani s pralnimi stroji na koncu lahko sklepamo, da gre za pralni stroj. Vmes pa v drugem oknu pregleduje ponudbo kuhinj. Identifikacija prepletenih sej ni enostavna, prav tako



Slika 1.4: Prepletena seja prikazuje uporabnika, ki naroča popravilo za pokvarjen aparat in obenem pregleduje ponudbo novih kuhinj.

pa takšnih sej ne moremo enostavno ločiti brez neke vrste kontekstne pomoči. Prepletene seje imajo negativen vpliv na kakovost podatkov, zato moramo zanje najti rešitev. Kakovost podatkov v spletnem posatkovnem skladišču namreč pomembno vpliva na kakovost analiz, ki uporabljajo kot vir spletno podatkovno skladišče. Na voljo imamo tri možnosti:

- se ne ukvarjamo s problemom prepletenih sej,
- prepletene seje enostavno zavržemo,
- poskušamo razplesti prepletene seje v njihove elementarne dele.

Pri prvi rešitvi prepleteno sejo upoštevamo kot eno samo sejo. Na ta način problema nismo rešili ampak se z njim samo ne ukvarjamo. Rešitev je slaba za kakovost podatkov. Primerna je samo za okolja, kjer je prepletenih sej izjemno malo ali pa ne predstavljajo opaznejšega vpliva na kakovost podatkov. Pri drugi možnosti prepletene seje izločimo. Če prepletene seje enostavno zavržemo, zavržemo z njimi tudi uporabne podatke o uporabi naše spletne strani. Takšne seje ponavadi generirajo naprednejši uporabniki ali pa uporabniki, ki večkrat uporabljajo naše spletne strani, katerih obnašanje je izjemno pomembno za nas. Na primer, pri sistemu e-Študent generirajo prepletene seje pomembni uporabniki, pri spletni trgovini EnaA pa kupci generirajo skoraj dvakrat več prepletenih sej kot ostali uporabniki. Najboljša možnost je razplet prepletenih sej, zato smo se odločili razviti metode za razpletanje.

1.4 Predstavitev teme disertacije

Razpletanje prepletenih sej predstavlja izhodišče za raziskovalno delo na doktorski disertaciji. Osredotočili se bomo na izboljšanje postopka predprocesiranja podatkov o klikotoku pred zapisom v spletno podatkovno skladišče. Proces razpletanja prepletenih sej je namreč ena izmed faz v procesu predprocesiranja. Podatke o klikotoku najprej očistimo in izvedemo proces osejevanja. Uporabniške seje, ki smo jih v postopku osejevanja restavrirali brez težav, imenujemo *čiste seje*. V postopku osejevanja zaznamo tudi *prepletene seje*. Tipično take seje vsebujejo več začetnih strani, ki se lahko pojavijo v notranjosti seje. Prepletenih sej ne znamo pravilno restavrirati brez neke vrste kontekstne pomoči ali postopka za razpletanje sej. Prepletene seje ločimo od čistih sej in nad njimi izvedemo dodaten postopek razpletanja sej. Razvili in preizkusili smo več postopkov razpletanje sej. Postopki razpletanja temeljijo na podlagi stohastičnih metod, ki so se izkazale kot zelo uporabne tudi pri reševanju nekaterih drugih problemov, povezanih s klikotokom. Razpletanje sej torej v samem bistvu temelji na verjetnosti prehodov uporabnikov med različnimi spletnimi stranmi. Zaradi splošnosti in enostavnosti smo se odločili uporabiti markovski model. Za učenje markovskega modela smo uporabili čiste seje. V ta namen lahko uporabimo čiste seje iz zadnjega polnjenja spletnega podatkovnega skladišča ali pa iz zadnjih nekaj polnjenj. Naučen markovski model služi kot osnova v postopkih razpletanja sej.

Reševanje problema razpletanja smo se lotili na dva načina. Prvi način je preprostejši in temelji na principu lokalnega razpletanja. Pri tem postopku za vsako stran prepletene seje preverimo, kakšna je verjetnost, da pripada eni izmed delno razpletenih sej. Stran prepletene seje vedno priredimo delno razpleteni seji, ki ji pripada z večjo verjetnostjo. Ker metoda temelji na požrešnem pristopu, ne zagotavlja razpletenih sej z največjo skupno verjetnostjo (in posledično pravilnostjo).

Drugi način razpletanja sej temelji na iskanju razpletenih sej, ki imajo največjo skupno verjetnost. Pri tem načinu uporabimo shemo za predstavljanje problemov, imenovano *prostor stanj*. Prostor stanj je graf, katerega vozlišča ustrezajo problemskim situacijam, danemu problemu pa pri tem ustreza iskanje poti v tem grafu. Za problem razpletanja sej vozlišče grafa predstavlja stanje delnega razpleta prepletene seje. Reševanje problemov zahteva preiskovanje grafa, pri tem pa naletimo na problem alternativ. Pri preiskovanju grafov v reševanju problemov se pogosto pojavi problem kombinatorične eksplozije zaradi naraščanja alternativ [10]. Ta problem rešuje hevristično preiskovanje, zato smo uporabili strategijo za preiskovanje alternativ z imenom hevristično iskanje po načelu najprej najboljši (ang. Best-First Heuristic Search). Pri tem pristopu za vsako vozlišče v prostoru stanj izračunamo *hevristično oceno*. Taka ocena nakazuje, kako obetavno je vozlišče glede doseganja ciljnega vozlišča. Iskanje vedno nadaljujemo v smeri najbolj obetavnega vozlišča v množici kandidatov. Pri izračunu hevristične ocene uporabimo naučen markovski model.

Osnovna izhodišča, ki smo jih postavili pri izdelavi rešitve:

- razvoj postopka razpletanja, ki bistveno pripomore k izboljšanju kakovosti razpletenih sej,

- postopek razpletanja mora biti splošno uporaben; delovati mora z različnimi viri klikotoka ne glede na to ali klikotok pripada novičarskemu portalu, spletnemu informacijskemu sistemu ali drugi spletni aplikaciji.

1.4.1 Struktura disertacije

Disertacija ima poleg uvoda in sklepnih ugotovitev še pet poglavij, predstavljenih v nadaljevanju.

V **drugem poglavju** predstavljamo področje podatkovnih skladišč in spletnih podatkovnih skladišč. Podan je pregled nad področjem, opisani so posamezni koncepti podatkovnega skladišča. Ogledamo si razliko med podatkovnim skladiščem kot shrambo podatkov in podatkovnim skladiščenjem kot procesom. Podrobneje so opisane značilnosti spletnega podatkovnega skladišča, s katerim se podrobneje ukvarjamo v doktorski disertaciji. V zadnjem delu poglavja je podan načrt in implementacija spletnega podatkovnega skladišča za spletni študijski informacijski sistem.

Tretje poglavje se posveča analizi problema prepletenih sej, pregledu dosedanjega dela na področju klikotoka in osvetlitvi problema s teoretične plati. V prvem delu je podrobneje predstavljen problem prepletenih sej, podani so razlogi za ločevanje takih sej. Nadalje je podrobneje opisan klikotok in dosedanja dela na tem področju. V zadnjem delu poglavja je prikazana kompleksnost problema razpletanja sej s kombinatoričnimi izračuni, podani so možni pristopi k reševanju problema razpletanja.

V **četrtem poglavju** opisujemo uporabljene pristope, testno metodologijo in razvite metode za razpletanje sej. V prvem delu je opisan markovski model in preiskovanje prostora stanj. Sledi opis testne metodologije in predstavitev postopkov vrednotenja kakovosti razpletanja. Drugi del poglavja je posvečen podrobnemu opisu obeh metod razpletanja: razpletanje z uporabo markovskega modela in razpletanje z uporabo preiskovanja prostora stanj.

Peto poglavje je namenjeno opisu testnih podatkov. Metode razpletanja smo testirali na umetno generiranih podatkih in dveh različnih realnih virih klikotoka: spletne trgovine in spletnega študijskega informacijskega sistema.

V **šestem poglavju** so podani eksperimentalni rezultati razpletanja z uporabo obeh razvitih metod. Prvi del poglavja je namenjen analizi problemskih situacij, drugi del pa rezultatom razpletanja sej. Začnemo z nekaterimi osnovnimi eksperimenti nad umetno množico podatkov, nadaljujemo pa z rezultati razpletanja nad realnimi množicami podatkov. Zadnji del poglavja je namenjen analizi rezultatov.

V **zaključnem poglavju** vse zaokrožimo v celovito sliko. Podamo sklepne ugotovitve, povzamemo najpomembnejše rezultate disertacije ter predlagamo nadaljnje raziskave.

1.5 Pričakovani prispevki k znanosti

Pričakovani prispevki k znanosti so:

- Pregled področij teme. Natančneje bomo pregledali in opisali področja spletnih podatkovnih skladišč. Podrobno bomo opisali trenutne rešitve in morebitne slabosti obstoječih rešitev.
- Analiza specifičnih težav pri zajemu in pripravi podatkov iz spletnih dnevnikov. Predstavili bomo obstoječe pristope in ideje, na katerih bo temeljila izvedba naših postopkov.
- Razvoj algoritmov za razpletanje prepletenih uporabniških sej. Pri tem se bomo oprli na markovski model in na strategijo reševanja problemov s preiskovanjem prostora stanj. Preučili bomo tudi možnost uporabe predznanja (ang. background knowledge) v markovskih modelih.
- Razvoj realističnega modela za vrednotenje delovanja zgornjega algoritma. Rezultati modela bodo zaradi narave podatkov v realističnih primerih težko ocenljivi. V ta namen moramo razviti postopek za vrednotenje rezultatov.
- Empirični preizkus izdelanih postopkov razpletanja v praksi. Postopke bomo empirično preverili na realnih podatkih spletne aplikacije in spletne trgovine.

POGLAVJE 2

Podatkovno skladišče

2.1 Uvod

Podatkovno skladiščenje je ena izmed najbolj pomembnih strateških pobud na področju informacijskih sistemov [30]. Igra ključno vlogo pri razumevanju obnašanja strank pri poslovanju, povezovanju poslovnih partnerjev v verigi dobaviteljev, implementaciji strategij upravljanja s strankami in podpori meritev uspešnosti posameznih poslovnih procesov. V sami osnovi je namen podatkovnega skladišča zagotavljanje namenskega vira podatkov za podporo v procesu odločanja [100].

V nasprotju s podatki razpršenimi v več podatkovnih bazah (transakcijskih sistemih), podatkovno skladišče združuje podatke na enem mestu. Vsi uporabniki in aplikacije dostopajo do istih podatkov. Vsi uporabniki vidijo "eno različico resnice". To omogoča boljšo kakovost podatkov in posledično kakovostnejše analize podatkov in proces odločanja.

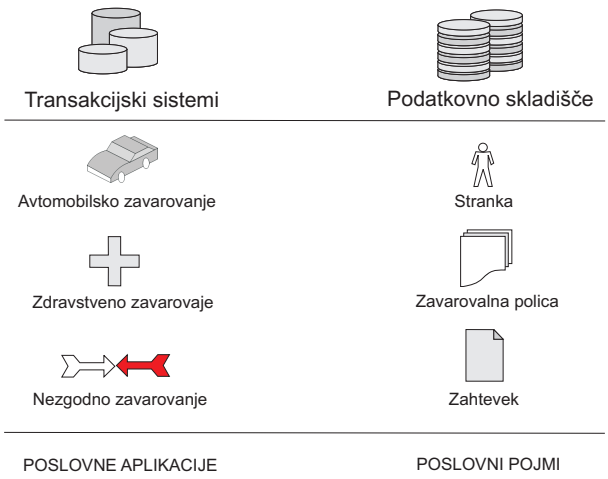
2.2 Osnovne definicije in koncepti

Podatkovno skladišče je integrirana zbirka podatkov, ki združuje podatke iz različnih virov in omogoča enostavno izvedbo poizvedovanj, potrebnih za izvajanje analiz in sprejemanje poslovnih odločitev [39, 16, 45]. Običajno združuje podatke iz več, porazdeljenih podatkovnih virov. Uporabniki in aplikacije dostopajo do tistih podatkov v podatkovnem skladišču, ki jih potrebujejo za reševanje nalog. Podatkovno skladišče predstavlja infrastrukturo podatkov v podjetju. Odpravlja oz. zmanjšuje razlog za neuspeh številnih aplikacij za podporo odločanju – uporabo nekakovostnih podatkov. Nudi temelje za učinkovit sistem za podporo odločanju v podjetjih, ki si hočejo ostati konkurenčna na trgu [18]. Podatkovno skladišče (PS) ima po

Inmonu [39] štiri temeljne značilnosti:

- je urejeno po predmetu obravnave,
- integrira podatke iz različnih virov,
- podatki so časovno odvisni,
- nespremenljivost.

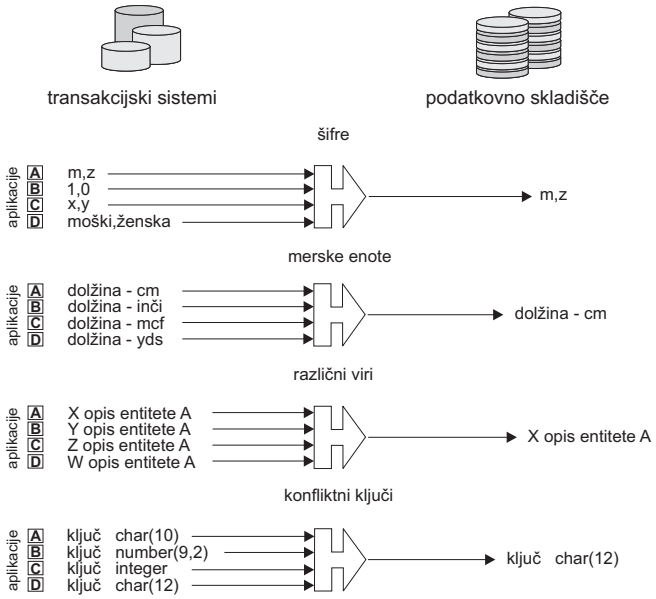
Urejenost po predmetu obravnave pomeni, da so podatki organizirani v smislu glavnih poslovnih pojmov, kot je stranka, izdelek, prodaja. Struktura podatkov v podatkovnem skladišču je različna od strukture v klasičnih transakcijskih sistemih, kjer so podatki organizirani glede na posamezne poslovne procese, ki se odvijajo ob poslovnih dogodkih, kot so naročilo, kontrola skladišča, prejem naročila itd. Na sliki 2.1 vidimo shematski prikaz razlik.



Slika 2.1: Urejenost po predmetu obravnave.

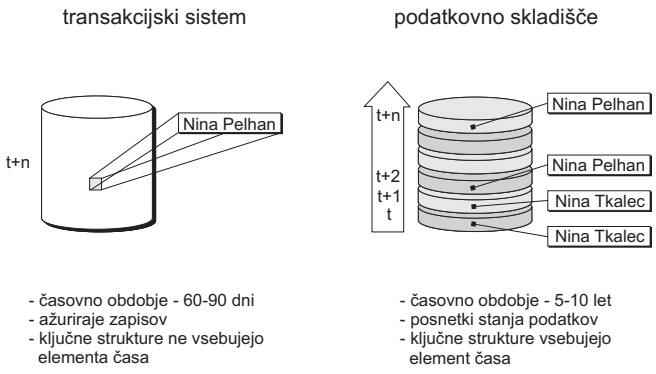
Integracija podatkov iz različnih virov pomeni, da se podatki zbirajo in shranjujejo v podatkovno skladišče iz številnih transakcijskih sistemov. Primer: podatke o strankah lahko dobivamo iz različnih notranjih (in zunanjih) transakcijskih sistemov. Podatke prečistimo in povežemo na podlagi šifre stranke tako, da dobimo vsesplošen in celovit pogled na stranko. Poenotenje se kaže v usklajevanju poslovnih pojmov, imen in domen atributov, ter uporabi enakih merskih enot, enakih opisov istih atributov ter ključev entitet. Shematično je integracija podatkov, pred zapisom v podatkovno skladišče, prikazana na sliki 2.2.

Časovna odvisnost pomeni, da podatkovno skladišče vsebuje podatke o daljšem časovnem obdobju. Podatkovno skladišče vsebuje zgodovinske podatke (to pomeni, da obravnava čas kot spremenljivko). Vsak podatek je veljaven v nekem točno določenem časovnem obdobju. V transakcijskih sistemih shranjujemo samo



Slika 2.2: Integracija podatkov iz različnih virov.

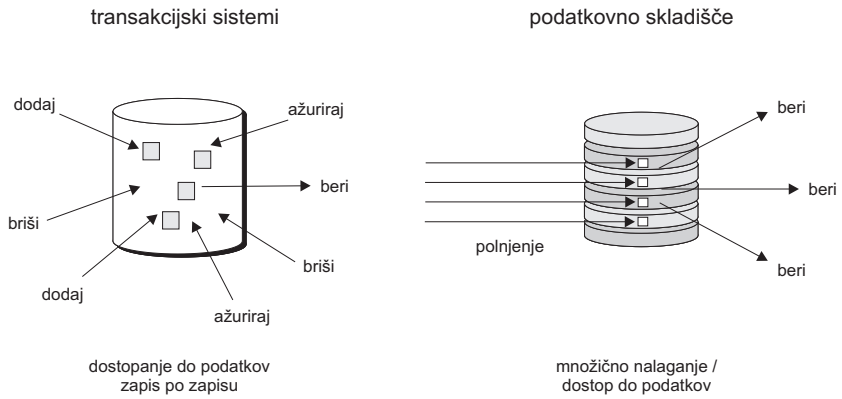
trenutno veljavno vrednost podatka, stara vrednost se ponavadi prepiše ali izbriše. V podatkovnem skladišču pa hranimo vrednost podatka na daljše časovno obdobje. Zgodovinske vrednosti podatkov potrebujemo za ugotavljanje odklonov in nepravilnosti, trendov in povezav na dolgi rok. Primerjavo med transakcijskim sistemom in podatkovnim skladiščem prikazuje slika 2.3.



Slika 2.3: Pomembna lastnost podatkovnega skladišča je časovna odvisnost.

Nespremenljivost pomeni, da se podatki v podatkovnem skladišču ne spreminjajo na način kot smo to vajeni iz transakcijskih sistemov – uporabniki podatkov ne morejo ažurirati ali brisati. Nespremenljivost zagotavlja, da vsi uporabniki kreirajo poročila nad isto množico podatkov. Seveda se podatki v podatkovnem skladišču spreminjajo, brišejo in dodajajo, vendar v okviru nadzorovanega procesa osveževa-

nja podatkovnega skladišča. Podatkovno skladišče v fazi osveževanja ni dostopno uporabnikom. Izven faze osveževanja pa je možno podatke samo brati. Nad transakcijskimi sistemi se v okviru izvajanja transakcij ali vzdrževalnih postopkov izvajajo postopki dodajanja, brisanja, ažuriranja in branja podatkov. Pri podatkovnem skladišču se praviloma izvajajo samo operacije osveževanja (dodajanja) in branja podatkov. Manj pogoste so operacije brisanja in ažuriranja (napačni podatki). Slika 2.4 prikazuje lastnost nespremenljivosti.



Slika 2.4: Lastnost nespremenljivosti.

Podatkovno skladišče vsebuje zgodovinske in agregirane podatke. Primer: podatkovno skladišče prodaje lahko vsebuje podatke o prodanih izdelkih, času prodaje, kraju in prodajalcu. Tipično je takšno podatkovno skladišče za velikostni red večje od transakcijske podatkovne baze in ne vsebuje podrobnih podatkov. Vsebuje pa pomembne podatke o podjetju, izdelkih in osebju [50].

2.2.1 Področno podatkovno skladišče

Področno podatkovno skladišče (ang. data mart) je podobno podatkovnemu skladišču s to razliko, da področno podatkovno skladišče hrani podatke za omejeno število poslovnih pojmov (ang. subject area), kot je npr. prodaja ali nabava [39]. Ker je manjše po obsegu in področju delovanja, vsebuje tudi manj podatkov in podpira manj aplikacij. Področna podatkovna skladišča so bodisi odvisna bodisi neodvisna [55].

- **Neodvisna** področna podatkovna skladišča so kreirana in osveževana neposredno iz izvornih transakcijskih sistemov. Podobno kot pri podatkovnem skladišču, se izvede faza izdelave izvlečkov, transformacije in polnjenja področnih podatkovnih skladišč.

- **Odvisno** področno podatkovno skladišče se kreira iz izvlečkov podatkov iz podatkovnega skladišča. Pred polnjenjem lahko seveda izvedemo dodatno

preoblikovanje podatkov in kreiranje agregiranih podatkov za hitrejše delovanje.

Neodvisno področno podatkovno skladišče je velikokrat trenutna rešitev nekega problema, ki na dolgi rok lahko povzroči pojavitev drugih težav. Na primer, nek oddelek potrebuje podatke za točno določeno poslovno aplikacijo in ima dovolj virov, da ustrezno področno podatkovno skladišče tudi zgradi. Zgrajeno področno podatkovno skladišče se lahko izkaže za uspešno, kljub temu pa lahko povzroči težave pozneje, ko organizacija poskuša kreirati globalno podatkovno skladišče na ravni organizacije.

Odvisno področno podatkovno skladišče zgradimo z namenom nuditi uporabnikom pogled na podatke po meri in potrebah uporabnikov. Na primer, podatke v zvezi s financami lahko preslikamo v odvisno področno podatkovno skladišče, ki ga bodo uporabljali finančni analitiki. Pogosto podatke v podatkovnem skladišču na ravni organizacije dopolnjujejo lokalna področna podatkovna skladišča, ki jih uporabljajo določene skupine uporabnikov. Tipično zagotavljajo hitrejše delovanje, kot vseobsegajoče podatkovno skladišče iz katerega izhajajo. S stališča organizacije so odvisna področna podatkovna skladišča veliko bolj zaželjena kot neodvisna, ker uporabljajo kot vir integriran podatkovni vir na ravni organizacije. Ta pristop omogoča manjša odstopanja med poročili različnih aplikacij. Vsi delajo z istimi podatki, čeprav se le ti nahajajo na različnih lokacijah.

2.2.2 Skladišče operativnih podatkov

Skladišče operativnih podatkov (ang. **operational data store** – ODS) združuje podatke iz številnih transakcijskih sistemov in predstavlja približek integriranih trenutno veljavnih podatkov iz transakcijskih sistemov. Proces osveževanja skladišča operativnih podatkov (ODS) je enak procesu za podatkovno skladišče z razliko, da v skladišču operativnih podatkov *ne hranimo zgodovinskih podatkov*. V ODS zelo redko najdemo podatke starejše od 30 ali več dni. Namen skladišča operativnih podatkov je zagotovitev integriranih podatkov za operativne namene. Pomagajo nam bolje spoznati in razumeti nek poslovni pojav. Nekatera podjetja gradijo ODS z namenom zaobiti realizacijo, velikokrat za podjetja preobsežnih, sistemov ERP (Enterprise Resource Planning). Eden izmed namenov sistema ERP je tudi integracija transakcijskih podatkov, kar pa lahko ceneje storimo z realizacijo ODS.

V zvezi z ODS se pojavlja tudi pojem *področnega skladišča operativnih podatkov* (ang. oper mart). Zgradi se jih takrat, ko se pojavi potreba po večdimenzionalni analizi podatkov v skladišču operativnih podatkov. Vir podatkov za področno skladišče operativnih podatkov je ODS. Njegove glavne značilnosti so:

- Predstavlja majhno podmnožico podatkov iz ODS in se uporablja za izdelavo taktičnih analiz.

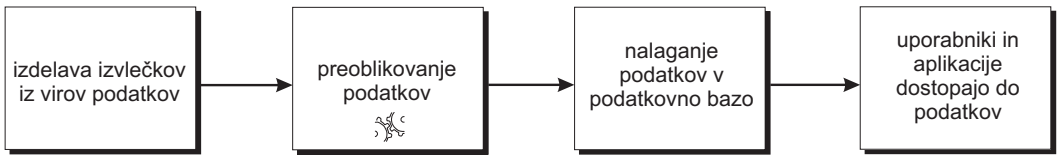
- Ažurira se v skladu z vsemi transakcijami, ki se izvedejo nad ODS.

- Podatki v področnem skladišču operativnih podatkov so shranjeni v večdimenzionalni obliki (zvezdna shema).
- Najbolj pomembno je, da je začasne narave in se ga poruši, ko ni več potrebno.

Kot primer navedimo slovensko zavarovalnico, ki hoče analizirati izpostavljenost tveganju, če bo tekoče poletje vroče s hudimi vremenskimi ujmami. Za izdelavo analiz potrebuje ažurne podatke o strankah v Sloveniji, njihove zavarovalne police in obseg zavarovanja kmetijskih površin, ki jih police pokrivajo. Podatke lahko kreiramo za ad hoc analize s pomočjo izdelave izvlečka iz ODS in kreiranje ustreznega področnega skladišča operativnih podatkov.

2.2.3 Podatkovno skladiščenje kot proces

Podatkovno skladišče v ožjem smislu je specializirana podatkovna baza z vsemi podatki. Podatkovno skladišče v širšem smislu pa predstavlja celoten proces, ki ga imenujemo *podatkovno skladiščenje*. Kot vidimo na sliki 2.5, podatkovno skladiščenje vsebuje obsežen rang aktivnosti od izdelave izvlečkov iz transakcijskih sistemov do uporabe podatkov v procesu odločanja. Proces skladiščenja podatkov obsega: izdelavo izvlečkov iz transakcijskih sistemov, transformacijo in preoblikovanje podatkov, nalaganje podatkov v podatkovno skladišče, ter dostop do podatkov s strani uporabnikov in aplikacij [100].

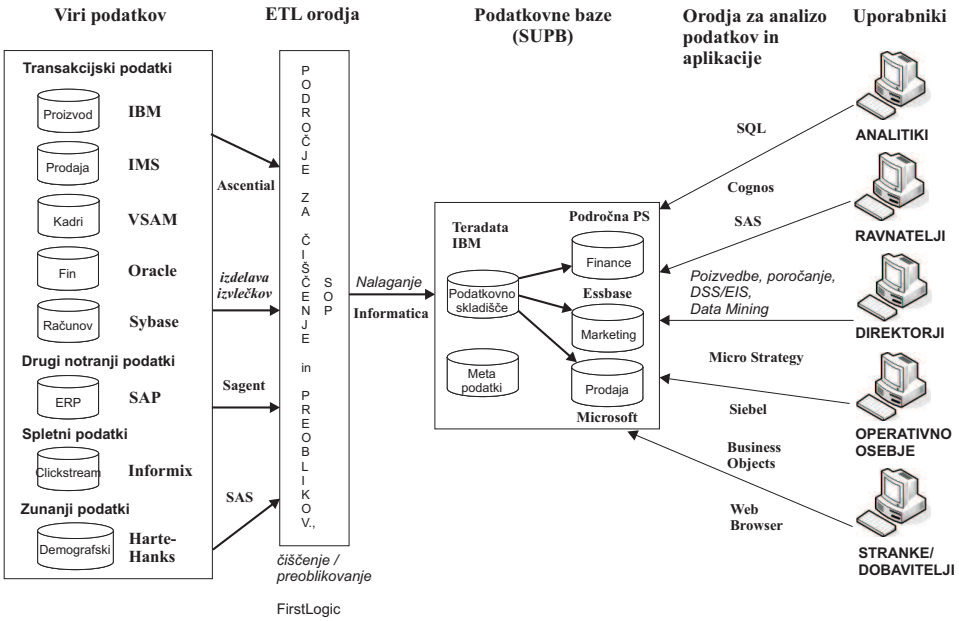


Slika 2.5: Podatkovno skladiščenje kot proces.

2.3 Arhitektura podatkovnega skladišča

Arhitektura podatkovnega skladišča je sestavljena iz komponent in povezav med komponentami. Vsaka komponenta služi določeni funkciji. Slika 2.6 prikazuje tipično splošno arhitekturo podatkovnega skladišča [100]. Na njej vidimo, da imamo štiri ločene komponente sistema: transakcijske sisteme kot vire podatkov, področje za čiščenje in preoblikovanje podatkov, podatkovne baze in predstavitveno plast z orodji za dostop do podatkov [16].

Leva stran slike 2.6 prikazuje različne podatkovne vire. Veliko podatkov prihaja iz transakcijskih sistemov, ki obsegajo različna poslovna področja (proizvodnja, nabava, prodaja). Podatkovno skladišče lahko polnimo s podatki iz ERP sistemov, kot



Slika 2.6: Splošna arhitektura podatkovnega skladišča.

so SAP ali PeopleSoft, polnimo ga lahko s podatki iz spletnih aplikacij (dnevniki spletnih strežnikov), seveda pa lahko podatkovno skladišče polnimo tudi z zunanjimi podatki. Podatkovni viri običajno uporabljajo različno strojno in programsko opremo in mešanico hierarhičnih, mrežnih in relacijskih podatkovnih modelov za shranjevanje podatkov. Veliko podatkovno skladišče ima lahko več kot 100 različnih virov podatkov.

Izvlečke podatkov naredimo iz virov podatkov s pomočjo lastnih aplikacij (napisanih v enem izmed višjih programskih jezikov) ali s pomočjo komercialne programske opreme za izločanje, preoblikovanje in nalaganje podatkov (ang. Extraction, Transformation, Loading – ETL). Podatke nato prenesemo v področje za čiščenje in preoblikovanje podatkov (ang. staging area), kjer jih ustrezno očistimo, preoblikujemo in prilagodimo. Da si poenostavimo proces čiščenja podatkov, lahko uporabimo namensko programsko opremo. Ponudniki namenske programske opreme omogočajo izdelavo postopka ETL, vendar lahko ne omogočajo nobenih optimizacijskih možnosti [87]. Prečiščene in obdelane podatke nato uvozimo v podatkovno skladišče. Običajno morajo biti ti procesi zaključeni v okviru določenega časovnega okna [87]. Postopek ETL ima zelo pomembno vlogo znotraj podatkovnega skladišča, ker predstavlja most med viri podatkov in podatkovnim skladiščem [109].

Podmnožico podatkov lahko uporabimo za izgradnjo odvisnih področnih podatkovnih skladišč, ki nudijo podporo specifičnim poslovnim uporabnikom (finančni analitiki, izvedenci za kontrolo kakovosti). Tipično odvisna področna podatkovna skladišča predstavljajo podatke v večdimenzionalni obliki.

Do podatkovnega skladišča dostopa veliko uporabnikov. Pri tem uporabljajo

orodja in aplikacije, ki so primerna za njihove potrebe. Napredni uporabniki (analitiki), ki razumejo podatkovni model podatkovnega skladišča in znajo pisati SQL poizvedbe, velikokrat dostopajo do skladišča s pomočjo svojih SQL stavkov. Veliko uporabnikov (analitiki, vodje) dostopa do podatkov s posebnimi aplikacijami, kot so Business Objects in Cognos. Ti izdelki imajo na voljo grafični vmesnik za izdelavo poizvedb, ki nato samodejno izdela potrebno SQL programsko kodo. Posebno izučeni analitiki lahko uporabljajo podatkovno skladišče za odkrivanje zakonitosti v podatkih (ang. data mining). Podatkovno skladišče pa lahko uporabljajo tudi aplikacije za podporo odločanju in v zadnjem času tudi okolja za spremljanje poslovnih procesov (ang. Business process management – BPM) [78]. Seveda lahko uporabljamo podatkovno skladišče tudi za programsko opremo, ki je namenjena reševanju specifičnih nalog.

2.4 Proces čiščenja in preoblikovanja podatkov

2.4.1 Uvod

V procesu izdelave izvlečkov, preoblikovanja in čiščenja podatkov – *proces ETL* podatke najprej pridobimo iz *podatkovnih virov*, jih preoblikujemo v obliko, ki je primerna za potrebe v procesu odločanja in jih nato zapišemo v ciljno podatkovno bazo. Kljub temu, da se proces ETL dogaja v ozadju podatkovnega skladišča, je to pomemben postopek. Ker je zapleten, časovno potraten, "umazan", in drag postopek mu velja posvetiti posebno pozornost. Veliko projektov izgradnje podatkovnih skladišč je bilo neuspešnih bodisi ker ni bilo možno izdelati ustreznih postopkov ETL bodisi zaradi težav s podatkovnimi viri, zaradi neprimerne tehnologije, nezadostnega strokovnega znanja na projektu in organizacijskih težav [104].

2.4.2 Viri podatkov

Podatkovno skladišče polnimo s podatki iz različnih virov. Najpogostejši viri podatkov so:

- **starejši informacijski sistemi** (npr. aplikacije v COBOL-u);
- **podatkovne baze** (Oracle, SQL Server, DB2);
- **ERP sistemi** - paketne aplikacije, ki podpirajo celoten poslovni proces organizacije;
- **podatki o obiskih spletnih strani** - z analizo teh podatkov poskušamo izvedeti čimveč o obiskovalcih spletnih strani, njihovih željah, udejstvovanjih in obnašanju. Temu viru podatkov bo v tem delu posvečeno največ pozornosti;

- **zunanji viri podatkov** - podatki, ki jih ne pridobimo iz informacijskega sistema organizacije ampak iz zunanjih virov. Večinoma so to nestruktuirani podatki kot so slike, večpredstavne vsebine itd.

Organizacije so v teku let razvile veliko število aplikacij, ki nudijo podporo poslovnim procesom v organizaciji. Veliko teh aplikacij je napisanih v COBOL-u in se uporabljajo še danes. Običajno so slabo dokumentirane, uporabljajo nejasna imena tabel in atributov tabel. Taki podatki so manj primerni za uporabo v procesu odločanja. Ker dostop do podatkov upočasnjuje obdelavo transakcij, uporaba takih podatkov za potrebe odločitvenega procesa nemalokrat privede do performančnih težav teh aplikacij.

Veliko podjetij je zamenjalo stare transakcijske sisteme (ang. legacy systems) z ERP (Enterprise Resource Planning) sistemi ponudnikov SAP, PeopleSoft, Oracle in J.D. Edwards. Sistemi ERP so pomemben vir podatkov za podporo procesu odločanja, toda pogosto hranijo podatke v zapletenih in lastniško zaščitenih podatkovnih strukturah. To pomeni, da je dostop do podatkov v sistemih ERP pogosto zahtevnejši. Zato so ponudniki sistemov ERP izdelali rešitve, ki omogočajo izdelavo področnih podatkovnih skladišč (npr. Business Warehouse podjetja SAP). Z uporabo programske opreme ponudnikov sistemov ERP izvajamo proces ETL nad ERP podatkovnimi datotekami in podatke prenesemo v področno podatkovno skladišče. Predhodno določene in ad hoc analize se izvajajo nad podatki v področnem podatkovnem skladišču. Pogosto imajo večji ponudniki podatkovnih baz na voljo postopke za izdelavo izvlečkov iz sistemov ERP.

Zelo pomemben vir podatkov predstavljajo tudi zapisi o obisku spletnih strani organizacije. Omenjeni podatki se nahajajo v dnevnikih spletnega strežnika in so običajno zelo obsežni. Podrobnejši opis tega vira podatkov najdemo v poglavju 2.10.

Nekatera podjetja vključujejo v podatkovna skladišča tudi zunanje vire podatkov, kot so: poslovni partnerji, stranke, vladne organizacije ter združbe, ki se ukvarjajo s pridobivanjem in prodajajo svojih podatkov.

2.4.3 Izdelava izvlečkov

Pri izdelavi izvlečkov iz podatkovnih virov imamo dve glavni možnosti:

1. **Izdelva lastnih orodij po meri**. Podjetja, ki dobro poznajo svoje transakcijske sisteme, dobro poznajo in razumejo zapletene povezave med podatki in imajo ustrezna tehnična znanja znotraj hiše, se velikokrat poskušajo izogniti nakupu drage programske opreme za podporo ETL. Programsko opremo ETL raje izdelajo sami po meri. Vseeno pa je trenutni trend v podjetjih, uporaba komercialnih ETL izdelkov.

2. **Nakup komercialnega izdeleka**. Specializirano programsko opremo ETL imajo na voljo večji ponudniki podatkovnih baz (npr. IBM, Teradata, Oracle) in podjetja, ki so specializirana na izdelavo programske opreme ETL (npr.

Ascential Software – DataStorage in SAS institute – SAS System). Pri specializiranih orodjih ETL, lahko razmeroma enostavno dostopamo do podatkov v SUPB različnih ponudnikov, izberemo tabele in atribute, ki jih bomo potrebovali, in prenesemo podatke na ponor (t.j. podatkovne baze, ki tvorijo skladišče ali področno podatkovno skladišče). Postopek lahko tudi sorazmeroma enostavno avtomatiziramo.

Orodja ETL postajajo vse bolj dodelana in vsebujejo vse več funkcij. Dobra orodja ETL omogočajo podjetju razumeti starejše transakcijske sisteme (ang. legacy systems) in nakazati težave, ki se bodo verjetno pojavile pri gradnji podatkovnega skladišča. Organizacija ima tako na voljo več informacij preden naredi znatno investicijo v tej smeri. Dobro orodje ETL omogoča:

- ocenitev kakovosti podatkov v tabelah,
- določiti ali so podatki v stolpcih konsistentni (atribut tabele se uporablja za shranjevanje enega dejstva),
- ugotavljanje povezav med atributi tabel in tabelami,
- predlaganje verjetnih primarnih in tujih ključev pri stikanju tabel in
- generiranje kode ETL.

2.4.4 Področje za čiščenje in preoblikovanje podatkov

Ko so narejeni izvlečki podatkov, jih navadno prenesemo v področje za *čiščenje in preoblikovanje podatkov* (ang. staging area), kjer jih preoblikujemo. Podatke lahko uvozimo s pomočjo osnovnih vmesnikov, tekstovnih datotek, povezav FTP in drugih postopkov. O področju za čiščenje in preoblikovanje podatkov lahko razmišljamo kot o procesu, ki uskladi podatke. Podatke pred prenosom v podatkovno skladišče ustrezno obdelamo. Uporabniki v tej fazi še nimajo dostopa do podatkov, ker podatki še niso pripravljeni za uporabo.

2.4.5 Preoblikovanje podatkov

Podatke iz transakcijskih sistemov preoblikujemo na različne načine, ki vključujejo: čiščenje, integracijo podatkov in ostale spremembe. Najprej podatke očistimo.

Čiščenje podatkov

Slaba kakovost podatkov v transakcijskih sistemih je rezultat slabih postopkov in glede tega lahko storimo zelo malo. Kljub temu, da bi organizacije morale stremeti k izboljšanju kakovosti podatkov v transakcijskih sistemih, pa se veliko projektov izgradnje podatkovnih skladišč sooča z nekakovostnimi podatki, vsaj na kratki rok. Watson [100] navaja glavne razloge za slabo kakovost podatkov: neustrezne vrednosti, nepopolni podatki, večnamenska polja, nejasni podatki, nasprotujoči si

podatki, kršenje poslovnih pravil, nepravilna uporaba vnosnih polj z naslovom, neunikatni identifikatorji težave z integriteto podatkov in ponovna uporaba opuščenih ključev. Pri čiščenju podatkov imamo na voljo več rešitev:

- zanašanje na osnovne funkcije, ki jo orodje ETL nudi za čiščenje,
- izdelava lastnih postopkov za čiščenje podatkov,
- uporaba posebnih namenskih orodij za čiščenje podatkov.

Namenska orodja se ponavadi uporabljajo za čiščenje imen in naslovov. Proces čiščenja podatkov lahko vidimo na sliki 2.7 [100]:

Slika 2.7: Proces čiščenja podatkov.

Povezovanje podatkov v celoto

Glavni namen podatkovnih skladišč je integracija podatkov iz več sistemov. Podatki združeni na enem mestu nudijo možnosti za kakovostnejše analize. Nekateri izmed podatkov morda v podjetju niso na voljo in jih je potrebno priskrbeti drugje. Skupna identifikacijska številka, kot je EMŠO ali davčna številka, v podatkovnem skladišču združuje veliko število različnih podatkov iz več transakcijskih sistemov. Tako podatkovno skladišče nudi možnosti za celovito razumevanje poslovnega procesa in njegovih dimenzij.

Ostala preoblikovanja

Druga preoblikovanja podatkov vključujejo zamenjavo vrednosti, izračunavanje izpeljanih vrednosti in izdelavo agregiranih podatkov. Vrednosti podatkov, ki se uporabljajo v transakcijskih sistemih, niso vedno razumljive uporabnikom podatkovnega skladišča. Na primer šifre A, B in C se v transakcijskem sistemu uporabljajo za označevanje neposredne, kataloške in internetne prodaje. Zamenjava šifer v skladišču z opisnimi imeni pripomore k večji razumljivosti.

Nekateri podatki, ki jih potrebujejo uporabniki, so samo rezultat izračuna, ki vsebuje enega ali več ostalih podatkov. Na primer, povprečje strankinih izdatkov je rezultat deljenja vsote vseh izdatkov deljeno s številom dni v določeni periodi. Izračunane podatke, ki jih uporabniki potrebujejo, lahko izračunamo preden zapišemo podatke v podatkovno skladišče. Izvajanje poizvedb nad podatkovnim skladiščem je posledično hitrejše. Običajno je tudi, da vnaprej izračunamo sešteta dejstva po

posameznih ravneh hierarhije. Tak postopek lahko znatno poveča zmogljivost sistema in skrajša odzivne čase, ker uporabnikom ni potrebno vedno znova seštevati podatkov.

Nalaganje podatkov

Ko so podatki preoblikovani, so pripravljeni za zapis v podatkovno skladišče. Ob prvem polnjenju se naložijo vsi podatki. Naknadna polnjenja lahko izvedemo na enega izmed dveh načinov. Prva možnost je vsakokratno nalagaje vseh podatkov v podatkovno skladišče (ang. bulk load). Pri tem pristopu se vsi podatki (stari in novi) napolnijo v podatkovno skladišče. Prednost tega pristopa je manj zapletena procesna logika, vendar postane zelo nepraktičen, ko se količina podatkov povečuje. Pogosteje uporabljen pristop je osveževanje podatkovnega skladišča samo z novo dodanimi podatki. Ta proces je bolj zahteven, ker moramo zajeti samo tiste podatke iz podatkovnih virov, ki so se spremenili. Ta proces v angleščini imenujemo *change data capture*.

Drugi problem, ki ga moramo razrešiti je, kako pogosto bomo dodajali podatke v podatkovno skladišče. Faktorji, ki vplivajo na odločitev so: poslovne potrebe po podatkih v podatkovnem skladišču in pa poslovni cikel, ki zagotavlja podatke. Uporabniki podatkovnega skladišča lahko potrebujejo dnevno, tedensko ali mesečno osveževanje skladišča z novimi podatki, odvisno od njihovih potreb. Večino poslovnih procesov ima naravni poslovni cikel. To pomeni, da generira podatke, ki jih lahko zapišemo v podatkovno skladišče na različnih točkah poslovnega cikla. Plačilne liste v podjetjih se tipično vodijo na mesečni osnovi. Posledično je najbolj verjetno, da bomo podatke dodajali v podatkovno skladišče na mesečni osnovi.

V zadnjem obdobju se vse bolj uveljavlja integracija podatkov v skoraj realnem času. Ustrezen pristop pri ažuriranju med transakcijskimi sistemi in podatkovnimi skladišči je torej postal ključen. V prid posodabljanju skladišča skoraj v realnemu času govori kar nekaj razlogov. Podatkovna skladišča se vedno več uporabljajo za podporo transakcijskim procesom, zato je zelo pomembno, da imamo v skladišču zadnje veljavne podatke. Drugi razlog je posledica odločitve, da se partnerjem omogoči vpogled v podatkovno skladišče. Poslovni partnerji pričakujejo, da so podatki čimbolj ažurni. Tretji razlog najdemo v multinacionalnih podjetjih, kjer ni primernega časa za osvežitev podatkov. Uporabniki po svetu potrebujejo dostop do skladišča po principu 24×7. Interval, v katerem bi bilo podatkovno skladišče neaktivno zaradi osveževanja (ang. load window), ni sprejemljiv. Italiano in sod. [40] predlagajo primer ogrodja za osveževanje podatkovnega skladišča v skoraj realnem času. Več na to temo najdemo v poglavju 2.8.2.

Shramba podatkov

Očiščene in preoblikovane podatke polnimo v podatkovno skladišče, v podatkovno bazo (ang. Data Store). Polnjenje lahko izvedemo na več načinov, v različne podatkovne baze:

- neposredno polnjenje področnih podatkovnih skladišč,
- polnjenje skladišča operativnih podatkov (ODS),
- polnjenje ODS, nato prenos podatkov iz ODS v globalno ali področna podatkovna skladišča,
- neposredno polnjenje globalnega podatkovnega skladišča.

Polnjenje podatkov v neodvisno področno podatkovno skladišče ni zaželjeno, ker težje omogoča analize na ravni podjetja, prej tudi pride do možnih nekonsistentnosti v poslovnih poročilih. Podatki se od tu prenesejo v podatkovno skladišče. Na nek način ODS postane vir podatkov za podatkovno skladišče, ki je ciljna podatkovna baza [39].

2.5 SUPB za podatkovno skladišče

Podatkovne baze za podatkovno skladišče lahko uporabljajo *relacijsko* ali *večdimenzionalno* tehnologijo. Pri uporabi relacijske tehnologije podatke shranjujemo v tabele z atributi in stolpci, kar je osnovni princip uporabe podatkovnih baz. Veliko realizacij podatkovnih skladišč uporablja samo relacijsko tehnologijo, ker ponuja robusten, zanesljiv in učinkovit pristop hranjenju velikih količin podatkov. Vendar pa tradicionalni relacijski strežniki ne procesirajo učinkovito zapletenih OLAP poizvedb. Ravno tako ne nudijo učinkovitega večdimenzijonalnega pogleda na podatke [17]. Razlog je v tem, da so prilagojeni za OLTP obdelave podatkov [92]. Zaradi teh razlogov postaja vse bolj privlačna uporaba tudi večdimenzionalnih podatkovnih baz.

Relacijske podatkovne baze zagotavljajo dimenzionalni pogled na podatke z uporabo modela zvezdne sheme. Model zvezdne sheme je sestavljen iz tabele dejstev na sredini in dimenzijskih tabel na stičišču zvezde. Zvezdna shema predstavlja denormaliziran podatkovni model. Slika 2.8 prikazuje primer zvezdne sheme za področje prodaje. Zvezdna shema ni edini možni logični model za večdimenzionalno podatkovno bazo [95], čeprav predstavlja zelo dober kompromis med preglednostjo, ki jo mora model zagotavljati, in stopnjo denormalizacije podatkovnega modela, ki je še dopustna s stališča redundance in zagotavljanja integritete podatkov. Nekoliko bolj podrobno je koncept zvezdne sheme predstavljen v podpoglavju 2.7 tega dela.

Pri uporabi večdimenzionalnih podatkovnih baz (ang. Multidimensional Databases – MDB) podatke shranjujemo v večdimenzionalne podatkovne strukture, imenovane kocke. V konceptualnem smislu so podatki v kockah urejeni v obliki zvezdne sheme. Uporabniku so na voljo dimenzije, predhodno določene hierarhije in nivoji. Vsako merljivo dejstvo je modelirano na preseku vseh smiselnih dimenzij. Prednost uporabe večdimenzionalnih podatkovnih baz je dejstvo, da ponujajo uporabnikom tak pogled na podatke, kot ga potrebujejo za učinkovito izdelavo numeričnih analiz [69].

Večdimenzijonalna podatkovna baza je vrsta podatkovne baze, prilagojena za podatkovno skladišče in aplikacije OLAP (online analytical processing). Večdimenzijonalna podatkovna baza je pogosto grajena tako, da uporablja kot vir podatkov obstoječo relacijsko podatkovno bazo. Zgrajene so tako, da optimizirajo odzivni čas za uporabniške večdimenzionalne analize. Za dostop do relacijskih podatkovnih se uporablja poizvedbe SQL. Večdimenzionalne podatkovne baze pa uporabljajo povpraševalne jezike, katerih sintaksa je blizu jeziku analitikov in računovodij (npr. Kakšen je delež prodaje mlečnih izdelkov po kategorijah v primerjavi z lanskim letom?).

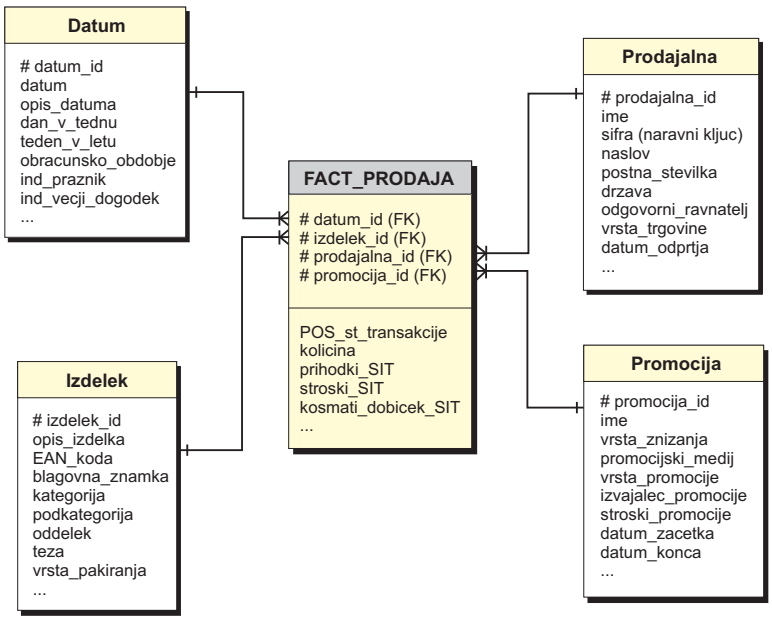
Večdimenzionalne podatkovne baze, kot so Oracle Essbase in Microsoft Analysis Server, predstavljajo dopolnilna orodja. Največkrat jih uporabljamo za hranjenje podatkov v odvisnih področnih podatkovnih skladiščih do katerih uporabniki dostopajo. Uporaba večdimenzionalnih podatkovnih baz predstavlja prednosti, kot je majhen odzivni čas, in tudi določene pomanjkljivosti. Te so: cena; večdimenzionalne podatkovne baze je potrebno vpeljati in se jih naučiti pravilno uporabljati in vzdrževati; tipično podpirajo manjše količine podatkov kot relacijske PB. Zadnja pomanjkljivost počasi izgineva, ker se kapaciteta večdimenzionalnih podatkovnih baz povečuje.

2.6 Dimenzijsko modeliranje

Analize podatkov zahtevajo drugačno organizacijo podatkov kot smo jo vajeni iz transakcijskih sistemov. V smislu podatkovnega modeliranja je koristno na podatkovno skladišče gledati kot na dimenzijski podatkovni model, ki je sestavljen iz centralne *tabele dejstev* in množice obkrožajočih *dimenzijskih tabel*. Vsaka dimenzijska tabela se nanaša na eno komponento ali *dimenzijo* tabele dejstev. V okviru relacijske podatkovne baze tabela dejstev vsebuje vse potrebne tuje ključe atributov dimenzij. Konceptualno to vodi do zvezdi podobni podatkovni strukturi, ki se imenuje *zvezdna shema*. Po Kimbalovem mnenju [45] dimenzijsko modeliranje dejansko določa entitetno-relacijski pristop modeliranja, ki je privzet pristop za modeliranje relacijskih podatkovnih baz. Nekoliko bolj podrobno je koncept zvezdne sheme predstavljen v naslednjem razdelku.

2.7 Zvezdna shema

Model zvezdne sheme je podoben zvezdi in je sestavljen iz tabele dejstev na sredini in dimenzijskih tabel na stičišču zvezde. Zvezdna shema predstavlja denormaliziran podatkovni model. Na sliki 2.8 vidimo primer zvezdne sheme za področje prodaje. Zvezdna shema ni edini možni model za večdimenzionalno podatkovno bazo, čeprav predstavlja zelo dober kompromis med transparentnostjo, ki jo mora model zagotavljati, in stopnjo denormalizacije podatkovnega modela, ki je še dopustna s stališča redundance in zagotavljanja integritete podatkov.



Slika 2.8: Primer zvezdne sheme za podatkovno skladišče prodaje na drobno.

Inmon [39] predlaga pristop, kjer so v globalnem podatkovnem skladišču podatki še vedno v normalizirani obliki, šele pri področnih podatkovnih skladiščih pride do reorganizacije v obliko zvezdne sheme. Kimball in sod. [45] predlagajo pristop, kjer se podatki nahajajo izključno v obliki zvezdne sheme. Kot smo že omenili, ime zvezdne sheme izhaja iz oblike podatkovnega modela, ki je sestavljen iz osrednje tabele, imenovane tabela dejstev. Na osrednjo tabelo je relacijsko povezanih več pomožnih tabel, imenovanih dimenzijske tabele.

Zvezdne sheme lahko preoblikujemo v *snežinkaste sheme* (ang. snowflake schema), ki nudijo podporo hierarhiji atributov tako, da dovolimo dimenzijskim tabelam, da imajo poddimenzijske tabele. Na primer, dimenzijska tabela `trgovina`, bi lahko imela poddimenzijsko tabelo `demografski podatki`, ki vsebuje demografske podatke o kraju, kjer se prodajelna nahaja. Poteka razprava o prednostih takega pristopa [50, 45, 78], ker njegova uporaba upočasnjuje obdelavo, toda v nekaterih primerih omogoča logično delitev podatkov, kot je v primeru demografske dimenzije [44].

Levene in sod.[50] trdi, da je snežinkasta shema primeren model za uporabo v podatkovnem skladišču. Dokler skrbimo za referenčno integriteto, ni težav pri polnjenju skladišča ter izvajanju poizvedb. Določi tudi formalizirano normalno obliko za podatkovna skladišča, imenovano *snowflake schema normal form* (SSNF), ki zajema intuicijo za snežinkasto shemo. Formalni opis določi na podlagi pomebnih konceptih teorije načrtovanja relacijskih baz. Pokaže, da ima normalna oblika za podatkovna skladišča SSNF več koristnih lastnosti. Ni jasno dokazano, če režijski stroški, ki nastanejo pri nadaljni cepitvi dimenzijskih tabel, prinesejo večjo dodano

vrednost kot je enostavnost zvezdne sheme. Prostorski prihranki, ki jih dobimo na ta način, so vsekakor majhni v primerjavi z velikostjo tabele dejstev.

2.7.1 Tabela dejstev

Tabela dejstev je glavna tabela v dimenzijskem modelu. Tu so shranjena merljiva dejstva o določenem predmetu poslovanja. Kot vidimo na sliki 2.8, si prizadevamo shraniti merljiva dejstva iz določenega poslovnega procesa v enem samem področnem podatkovnem skladišču. Ker merljiva dejstva zavzemajo največji del poljubnega področnega podatkovnega skladišča, se izogibamo podvajanju istih merljivih dejstev po različnih področnih podatkovnih skladiščih.

Izraz *dejstvo* (ang. measurable fact) označuje poslovno merilo. Predstavljajmo si prodajo v trgovini, kjer kontroliramo količino prodanih izdelkov, na določen dan v določeni trgovini. Meritev poteka na preseku vseh štirih opazovanih dimenzij (čas, izdelek, trgovina, promocija). Seznam dimenzij določa *zrno* (ang. grain) tabele dejstev in predstavlja okvir meritve. Meritev je torej vrstica v tabeli dejstev. Zrno tabele dejstev predstavlja najbolj natančno raven podatkov, ki je na voljo v dimenzijskem modelu. Zrno nudi odgovor na vprašanje: kako opišemo posamezno vrstico v tabeli dejstev. Primeri zrna tabele dejstev so:

- Individualna postavka računa stranke v trgovini.
- Individualni mesečni posnetek stanja na računu v banki.
- Dogodek vpisa študenta na fakulteto.
- Dnevni posnetek stanja zalog v blagovnem skladišču.
- Dostop do spletne strani na spletnem strežniku podjetja.

Vse meritve v tabeli dejstev morajo imeti enako zrnatost. Najbolj uporabna dejstva so numerična in seštevna (jih lahko seštevamo). Seštevnost merljivih dejstev je odločilnega pomena za podatkovno skladišče. Aplikacije zelo redko prikazujejo podrobne podatke, večinoma so njihova poročila sestavljena iz velikega števila agregiranih podatkov. Polseštevna (ang. semiadditive) so tista dejstva, ki jih lahko seštevamo samo po določenih dimenzijah, neseštevnih (ang. nonadditive) pa sploh ne moremo seštevati. Primer polseštevnega dejstva je stanje na računu, kjer lahko naredimo seštevek po vseh strankah, ni pa smiselno seštevanje po času. Primer neseštevnega dejstva pa je ocena kreditnega tveganja stranke. Pri uporabi neseštevnih dejstev smo v poročilih omejeni na preštevanje zapisov in računanje povprečne vrednosti. Teoretično je možno, da je merljivo dejstvo tekstovno, vendar le izjemoma. Če je le mogoče tekstovne atribute prenesemo v eno izmed dimenzij. Če je tekstovno dejstvo enolično za vsako vrstico v tabeli dejstev, potem spada v tabelo dejstev. Pravo tekstovno merljivo dejstvo redko srečamo, ker zaradi nepredvidljive vrednosti (poljuben tekst) zelo otežkoča kakršnekoli analize.

Tabela dejstev ima ponavadi manjše število stolpcev. Tipično pa zavzema okrog 90 odstotkov prostora, ki ga zajema podatkovno skladišče [45]. Vsaka tabela dejstev ima primarni ključ, ki je sestavljen iz podmnožice tujih ključev. Tak ključ imenujemo *sestavljen primarni ključ*. Vse ostale tabele so dimenzijske tabele.

Samo podmnožica komponent sestavljenega ključa je tipično potrebna za zagotovitev enoličnosti. Samo nekaj izmed dimenzij torej lahko že enolično določa vsak zapis. Ostale dimenzije sicer sestavljajo sestavljeni ključ, vendar ne pripomorejo k enoličnosti. Tabela dejstve tipično tudi ne potrebuje stolpca, ki bi predstavljal šifro vrstice (ROWID). Tabela dejstev bi bila večja, kakih posebnih koristi pa ne bi bilo.

2.7.2 Dimenzijske tabele oziroma dimenzije

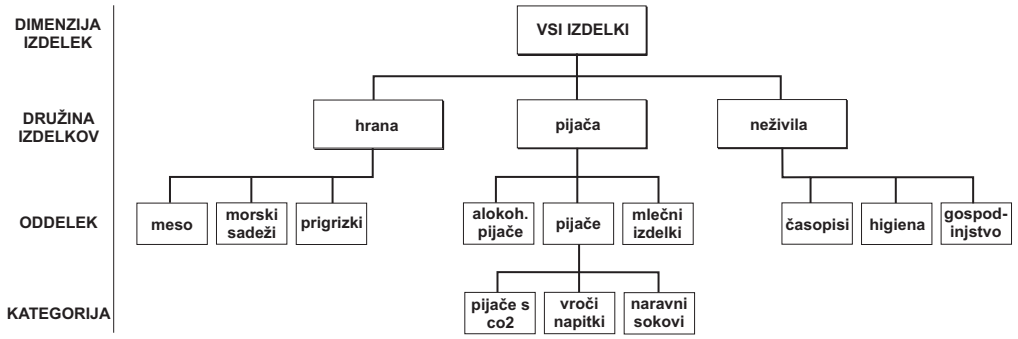
Dimenzijske tabele so nerazdružljivo povezane s tabelo dejstev. Sestavljene so iz ključa in atributov, ki nosijo tekstovno informacijo oz. opis neke dimenzije poslovanja. V dobro zastavljenem dimenzijskem modelu imajo dimenzije veliko atributov. Dimenzijske tabele imajo lahko 50 do 100 atributov. Prizadevati si moramo, da vključimo čimveč kakovostnih tekstovnih atributov, ki opisujejo dimenzije in hierarhijo, kot je mogoče. Atribute dimenzije uporabljamo za omejitve nad poročili, za grupiranje in naslove posameznih postavk v poročilih.

Dimenzijske tabele imajo v podatkovnem skladišču izjemno pomembno vlogo. Dobro zastavljene dimenzije z veliko atributi pomagajo pri kakovostni uporabi podatkovnega skladišča. Predstavljajo uporabniški vmesnik do podatkovnega skladišča. Najboljši atributi dimenzij so tekstovni in diskretni. Raje kot nejasne kratice, ki ne pripomorejo k razumevanju dimenzije, morajo biti sestavljeni iz celotnih besed.

Dimenzijske tabele običajno hierarhično urejajo določene poslovne pojme. Tipičen primer dimenzije s pripadajočo naravno hierarhijo je dimenzija izdelek. Izdelek pripada določeni družini izdelkov, ta spada v nek oddelek in nadalje v kategorijo. Vpeljava hierarhij je pomembna, ker omogoča izdelavo analiz na različnih ravneh podrobnosti. V primeru na sliki 2.9 omogoča spremljanje prodaje po družinah izdelkov, lahko pa gremo podrobneje po oddelkih ali še podrobneje po kategorijah izdelkov. Temu konceptu pravimo *vrtanje v globino*. Orodja za realizacijo poizvedb morajo omogočati enostaven prehod iz manj podrobnih analiz na bolj podrobne. Temu pravimo vrtanje navzdol oz. vrtanje navzgor.

Dimenzijska tabela običajno vsebuje podatke iz več tabel transakcijskega sistema. Redundanca na porabo prostora, zaradi majhnosti dimenzijskih tabel (tipično 10 odstotkov velikosti skladišča), nima večjega vpliva.

Poseben primer dimenzije je *časovna dimenzija*, ki nima fizičnega podatkovnega vira, ampak se generira programsko. Poseben primer dimenzijske tabele je tudi t.i. *izrojena dimenzija* (ang. degenerate dimension). To je dimenzijska tabela, ki nima nobenega atributa razen ključa. Vse ostale atribute smo prerazporedili po ostalih dimenzijah. Ključ je sestavni del sestavljenega ključa tabele dejstev, pripadajoča dimenzijska tabela pa ne obstaja. Izrojene dimenzije ponavadi uporabimo za grupiranje podatkov po skupinah.



Slika 2.9: Primer hierarhije za dimenzijo *izdelek*.

2.8 Trendi v skladiščenju podatkov

2.8.1 Osnova poslovnemu obveščanju

Skladiščenje podatkov je postalo osnova za realizacijo sistemov poslovnega obveščanja (ang. Business Intelligence – BI), le-to pa je v zadnjih letih glavna prioriteta ravnateljev informatike (ang. Chief information officer – CIO). Iz raziskav skupine Gartner Group o pomena projektov BI v zadnjih treh letih je razvidno, da se za ravnatelje informatike ti projekti nahajajo v samem vrhu pomembnosti [101, 96]. Priprava in zapis podatkov v podatkovno skladišče je še vedno eden izmed izzivov poslovnega obveščanja, ki zahteva 80% časa in napora in je vir 50% nepričakovanih stroškov na projektih [101].

2.8.2 Skladiščenje v realnem času

Zahteva po svežih podatkih v podatkovnem skladišču je bila vedno zelo zaželjena za skupino uporabnikov podatkovnega skladišča. Običajno poteka (klasično) osveževanje v paketnem načinu, v nočnih urah, ko je skladišče v fazi mirovanja. Ažurirani podatki iz virov podatkov se med delovnim časom hranijo v medpomnilniku. V postopku ETL, ko se podatki najprej preoblikujejo in očistijo, se podatki iz medpomnilnika prenesejo v skladišče. Podatkovnega skladišča med osveževanjem ni možno uporabljati za izvajanje poizvedb. Množica navedenih postopkov se izvaja v ponoči zato, da se izognemo dodatnemu obremenjevanju transakcijskih sistemov s postopki polnjenja [41].

Ker uporabniki želijo vse večjo ažurnost podatkov v skladišču in transakcijskem sistemu, se pojavljajo novi načini za učinkovitejše polnjenje in rabo podatkovnega skladišča. Relativno nov pristop, ki se je pojavil na področju podatkovnega skladišča je *aktivno podatkovno skladiščenje* (ang. Active data warehousing), pri katerem se podatkovno skladišče osvežuje sprotno, da zadosti visokim zahtevam uporabnikov po svežih podatkih. Na ta način je dosežena večja konsistentnost med podatki v transakcijskem sistemu in podatkovnem skladišču [74]. V povezavi z aktivnim po-

datkovnim skladiščenjem se zato uporablja tudi izraz *skladiščenje v realnem času* (ang. Real-time data warehousing) [78].

Skladiščenje v realnem času teži k zmanjševanju časa, ki je potreben, da lahko sprejemamo poslovne odločitve. V idealnem primeru ne bi smelo biti nobene zakasnitve med vzrokom in posledicami nad poslovnimi odločitvami. Skladiščenje v realnem času omogoča analiziranje podatkov podjetja nad vsemi podatkovnimi viri. Omogoča hitro odzivanje na spremembe v transakcijskih sistemih in tako učinkovito premošča vrzel med sistemi za podporo odločanju in poslovnimi procesi. Poslovne zahteve med podjetji se razlikujejo, zahteve po informacijah pa so podobne – povezane, točne, pravočasne, natančne in nemudoma dostopne [11].

Podatkovna skladišča predstavljajo celovit pogled na podatke podjetja in nudijo najprimernejšo podlago za izgradnjo okolja za *spremljanje poslovnih procesov* (ang. Business Process Monitoring – BPM). Izgradnja BPM nad podatkovnim skladiščem ima velik vpliv na izgradnjo in modeliranje skladišča, ker lahko vsebuje komponente, ki jih ne najdemo v klasičnem podatkovnem skladišču. Tak primer predstavlja nov način polnjenja PS s pomočjo podatkovnih tokov. BPM zahteva podatke v realnem času kar pomeni spremenjen pristop k obstoječim postopkom ETL, ki niso več ustrezni. Za doseganje zadostnih performančnih zahtev je potrebna uporaba naprednejših logičnih modelov za shranjevanje podatkov [78]. Sprememba paradigme polnjenja povzroči pojavitev več izzivov pri implementaciji procesa (postopka) ETL, ker mora postopek ETL potekati neprestano, ko potekajo spremembe nad transakcijskim sistemom [74]. Zahteve po sprotnem polnjenju podatkovnih skladišč predstavljajo izziv tudi zaradi: (i) Virov podatkov ponavadi ne moremo dodatno obremeniti z dodatnimi nalogami razpošiljanja podatkov proti skladišču. (ii) Implementacija aktivnega polnjenja podatkov v povezavi z podedovanimi transakcijskimi sistemi (ang. legacy system) ni trivialna naloga. Glavni izzivi pri načrtovanju so [78]:

- Pravočasnost: model mora omogočati pravočasno obdelavo podatkov. Informacije morajo biti na voljo takrat, ko so potrebne v procesu za podporo odločanju, in ne pozneje. V fazi načrtovanja moramo ugotoviti, kaj pomeni ”pravočasno” za določen poslovni proces.

- Ključni kazalniki zmogljivosti (ang. Key Performance Indicators – KPI): Pri spremljanju poslovnih procesov so lahko v veliko pomoč nadzorne plošče in mehanizmi sklepanja, ki odločevalcu nudi natančno in pravočasno sliko stanja poslovanja. Potrebne so torej tehnike za izgradnjo kazalnikov KPI in poslovnih pravil.

- Načrt procesov: Spremljanje poslovnih procesov zahteva tudi razumevanje poslovnih procesov in njihovih povezav, da lahko najdemo pomembne KPI in pravila ter podatke na osnovi katerih jih pridobimo.

Izziv postane še večji, če vemo, da ne moremo občutno spreminjati konfiguracije transakcijskih sistemov zaradi omejenega časa njihove neaktivnosti in stroškov

povezanimi z vzdrževanjem. Karkasidis in sod. [41] so predlagali ogrodje za implementacijo aktivnega podatkovnega skladišča s cilji: (i)izvesti minimalne spremembe na viru podatkov, (ii) minimalna dodatna obremenitev virov podatkov za potrebe aktivnega skladiščenja in (iii) premočrten nadzor okolja sistema.

Pri sprotnem polnjenju podatkovnega skladišča je eden izmed problemov tudi zamenjava ključa zapisa iz transakcijskega sistema s ključem, ki se uporablja v podatkovnem skladišču (ang. surrogate key). Zamenjava mora potekati hitro. Proces zamenjave se izvaja z omejenimi viri med hitrim tokom podatkov iz transakcijskega sistema S in počasnejšimi podatki v skladišču R (disk). Polyzotis in sod. [74] predlagajo nov način, kako ta postopek izvesti čimhitreje z uporabo *mesh join* operatorja. Zhu in sod. [109] pa so predstavili polnjenje podatkovnega skladišča v realnem času z uporabo storitveno usmerjene arhitekture (ang. Service Oriented Architecture – SOA).

2.8.3 Skladiščenje podatkov s spleta

V zadnjem desetletju so se pojavila podatkovna skladišča, ki zbirajo in hranijo podatke s svetovnega spleta (www). Zbiranje podatkov na spletu prinaša veliko težav, predvsem zaradi slabše struktuiranosti podatkov, pomanjkanje nadzora nad viri podatkov in nezmožnost ažurnega ugotavljanja sprememb. Podatki na spletnih straneh se nahajajo v obliki formatu HTML, ki nudi samo delno podporo strukturi podatkov. Glavni izziv na tem področju je, kako integrirati različne spletne vire in kako avtomatizirati postopek polnjenja v primeru, ko se večina podatkov za skladišče, nahaja na svetovnem spletu. Nekaj rešitev na tem področju je bilo narejenih z uporabo standarda XML [99]. Pri nekaterih drugih pristopih pa se razvoj skladišča podatkov za splet temelji na podlagi pogostih uporabniških poizvedb in na podlagi kakovosti podatkov [78].

2.8.4 Porazdeljena podatkovna skladišča

Podobno kot pri porazdeljenih podatkovnih bazah, je treba tudi pri porazdeljenih podatkovnih skladiščih pri načrtovanju vpeljati dve novi fazi: načrt porazdelitve sistema po arhitekturni in fizični plati. Z arhitekturnega stališča moramo sprejeti odločitve o uporabi ustrezne metode porazdeljevanja, ki bolj ustreza dejanskim zahtevam (P2P, grid). Določiti moramo tudi način vzpostavitve podatkovnega skladišča na izbrani podlagi. Primer dejanske implementacije nad različnimi paradigmami, najdemo v [1, 73]. Po fizični plati moramo določiti kako porazdeliti podatkovno skladišče po posameznih vozliščih, da dosežemo čimvečjo dosegljivost podatkov, omogočimo čimvečjo možnost paralelnih obdelav podatkov in seveda čimboljših zmogljivosti.

Uporaba porazdeljevanja je še posebej koristna takrat, ko pogosto dodajamo nova področna podatkovna skladišča zaradi nakupov ali zlivanja več podjetij.

2.9 Spletna podatkovna skladišča

2.9.1 Uvod

Svetovni splet (WWW) se v zadnjih desetletjih razvija z izjemno hitrostjo. Povečuje se število njegovih uporabnikov [63] in kompleksnost spletnih strani [20]. Vsak dan se na spletu na novo pojavi 20 miljonov novih spletnih dokumentov [90]. Vse več organizacij uporablja splet kot medij, preko katerega dostopajo do njihovih informacijskih sistemov (IS) in aplikacij [54]. Z razvojem spleta se je pojavil nov pojem: *spletna aplikacija* [53]. To je aplikacija, ki za uporabniški vmesnik uporablja splet in operacijski sistem osebnega računalnika. Nenehna rast uporabe spleta je pospešila razvoj novih in novih spletnih aplikacij, ki omogočajo povezave med strankami in poslovnimi partnerji [34].

Z večanjem števila spletnih strani in dokumentov imajo obstoječe spletne strani vedno večje težave s pridobivanjem novih uporabnikov in ohranjanjem obstoječih. Iz danih razmer lahko sklepamo, da bodo na dolgi rok uspešne samo tiste spletne strani, ki bodo razumele potrebe svojih uporabnikov. Analiziranje *obnašanja uporabnikov* (ang. user behavior) je postal torej pomemben del analize podatkov o nekem spletnem mestu. Spremljanje obnašanja uporabnikov nam pomaga pri razumevanju njihovih potreb in želja ter omogoča prilagoditev sistema uporabnikom na podlagi njihovega preteklega obnašanja. Poznavanje obnašanja uporabnikov omogoča poleg sprememb sistema tudi: preverjanje skladnosti sistema v skladi z začetnimi specifikacijami, omogočanje strategije prilagoditev sistema posameznemu uporabniku (ang. personalization), povečanje zmogljivosti sistema [68], pomoč pri trženjskih prijemih in zaznavanje poslovnih priložnosti, ki bi sicer ostale neopažene ter povečanje varnosti sistema [54]. Zaporedje klikov, ki jih uporabnik naredi med sprehajanjem po nekem spletnem mestu, imenujemo *klikotok* (ang. clickstream) [91].

Če hočemo podatke o obnašanju uporabnikov pridobiti in analizirati, jih moramo preoblikovati in shraniti v primerni obliki. Uporabimo *spletno podatkovno skladišče*, ki hrani podatke o klikotoku in druge kontekstne podatke. To je podatkovno skladišče, ki kot vir uporablja podatke o klikih uporabnikov na nekem spletnem mestu. S pojavitvijo spletnih aplikacij se je potreba po analiziranju obnašanja uporabnikov in posledično uporaba spletnih podatkovnih skladišč povečala. Slika 2.10 prikazuje arhitekturo spletnega podatkovnega skladišča (SPS). Arhitekura je na videz dokaj podobna klasičnemu podatkovnemu skladišču.

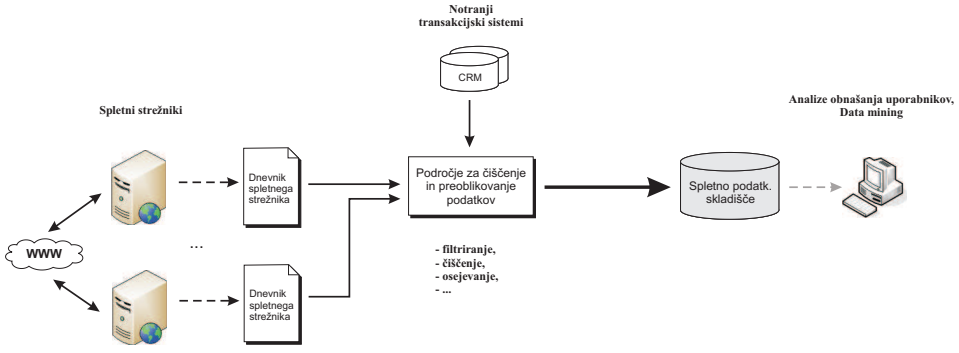
Leva stran slike prikazuje vire podatkov za spletno podatkovno skladišče. Glavni vir predstavlja klikotok, ki ga najpogosteje pridobimo iz dnevniških datotek enega ali več povezanih ali neodvisnih spletnih strežnikov. Druge možnosti za pridobitev podatkov o klikotoku so še: (i) ponudnik internetnih storitev, (ii) specializirana podjetja, ki smo jih najeli z namenom, da izvajajo kontrolo prometa na določenih spletnih mestih, (iii) podatke lahko odkupimo od komercialnih spletnih strani, kar je tudi že precej dobro utečena praksa.

Srednji del predstavlja postopek čiščenja in preoblikovanje podatkov (postopek

ETL). Zaradi slabše strukture podatkov in pomanjkanja določenih kontekstnih podatkov je postopek čiščenja in preoblikovanja podatkov pri SPS še posebej pomemben. Imenujemo ga tudi *poobdelava podatkov klikotoka* (ang. post-processing). V veliko primerih lahko za postopek poobdelave podatkov klikotoka uporabimo postopke, ki smo jih uporabili pri drugih podatkovnih skladiščih. Kljub temu pa obstaja nekaj pomembnih razlik, ki se jih moramo zavedati. V okviru aplikacije, ki izvaja poprocesiranje, moramo poskrbeti za [43]:

- **Filtriranje neuporabnih podatkov.** Podatke moramo ustrezno združiti in zavreči zapise, ki jih ne bomo prenašali v podatkovno skladišče. Čimbolj moramo zmanjšati obseg vhodnih podatkov tako, da še ustrezajo zrnu tabele dejstev in podatki še ostanejo celoviti.
- **Idenfifikacija sej.** Zapisom v dnevniški datoteki dodamo ustrezno šifro seje in preverimo, da si dogodki v seji sledijo v usteznem časovnem zaporedju.
- **Identifikacija uporabnikov.** Uporabnike iz dnevniških datotek poskušamo povezati z obstoječimi šiframi uporabnikov. Če to ni možno (neznan uporabnik), dodelimo uporabniku anonimen ID.
- **Identifikacija odjemalcev (ang. hosts).** Ugotovimo identiteto naslovov IP za odjemalce in vire zahtev (ang. referrer). Obdržimo podatke o državi vira zahteve in osnovne podatke o domeni (com, edu).
- **Podatke spravimo na skupen format zapisa.** Podatke o klikotoku zapišemo v obliki, ki je natančno definirana in ustreza aplikaciji za vnos podatkov v podatkovno skladišče.
- **Ugotavljanje podatkov za dimenzijo** *dogodek*.

Desna stran slike predstavlja podatkovno skladišče z eno ali več povezanimi zvezdnimi shemami. Do njih dostopamo z orodji za ugotavljanje zakonitosti v podatkih, vršimo poizvedbe in izvajamo analize.



Slika 2.10: Arhitektura spletnega podatkovnega skladišča.

2.10 Klikotok

Kot smo že navedli, je klikotok zaporedje klikov, ki jih uporabnik naredi med sprehajanjem po spletnem mestu. Podatke o klikotoku lahko pridobimo iz dveh virov: (i) klikotok, ki se nahaja na strežniški strani, (ii) podatki o klikotoku, ki se nahajajo na odjemalčevi strani. Če se med odjemalcem in strežnikom nahajajo tudi kaki posredniki, kot so proxy strežniki in analizatorji paketov (ang. packet sniffers), ti posredniki lahko predstavljajo koristen vir podatkov o klikotoku [71, 89].

Podatke na strežniški strani pridobimo na spletnem strežniku spletnega mesta. Podatki, ki so na voljo na strežniški strani so: dnevniški zapisi spletnega strežnika, piškotki, neposredni uporabniški vnosi in podatki pridobljeni iz zunanjih virov. Najpogosteje se ti podatki nahajajo v obliki različnih vrst dnevniških podatkov, generiranih s strani spletnega strežnika. Ti zapisujejo dostope do svojih virov v eno ali več dnevniških datotek (ang. HTTP log file). Vsak spletni strežnik ima to lastnost v takšni ali drugačni obliki vgrajeno v arhitekturo. Podatki o klikotoku, pridobljeni na strežniški strani, imajo določene prednosti pred podatki o klikotoku, pridobljenimi s strani odjemalca. Podatke lahko zbiramo za vse uporabnike, ki dostopajo do spletnega mesta brez potrebe po namestitvi dodatne programske opreme na odjemalčevi strani. Največja pomanjkljivost tega pristopa pa je, da smo pri interakciji med uporabnikom in strežnikom omejeni na nivo spletnega vira (npr. spletna stran). Zaradi narave protokola HTTP nimamo neposredne povezave med zahtevanimi viri. Poleg polja `referrer` ki pove, s katerega vira je uporabnik prišel, nimamo podatka o tem, kako uporabnik prehaja med različnimi spletnimi stranmi, kakšni so njegovi interesi v splošnem in kako je spletna stran pozicionirana na njegovi poti po spletnem mestu.

Podatki o klikotoku so običajno veliki, slabše struktuirani in prikazujejo samo delno sliko uporabnikove aktivnosti. V dnevniku spletnega strežnika na primer, ne najdemo sledu o aktivnostih, ki so posledica predpomnilnika brskalnika ali omrežja. Klik uporabnika na gumb `Nazaj` v brskalniku ne bo imel pripadajočega zapisa v dnevniku spletnega strežnika [47].

Podatki o klikotoku so lahko zapisani v različnih formatih, ki imajo opcijske podatkovne komponente. Zaradi porazdeljene narave spleta se podatki o obisku spletnih strani lahko zbirajo na več strežnikih hkrati, čeprav uporabnik misli, da deska na enem samem strežniku. Zaradi tega se pojavi tudi težava s sinhronizacijo in združevanjem podatkov iz več dnevniških datotek. Srednje obremenjen spletni strežnik lahko servisira več sto strani v eni sami sekundi. Zelo malo verjetno je namreč, da bi bile ure na različnih strežnikih usklajene na stotinko sekunde natančno.

2.10.1 Komunikacija spletni odjemalec / strežnik

Za lažje razumevanje spletnih dnevnikov, si poglejmo osnovno tehnologijo delovanja spleta. Razumevanje komunikacije odjemalca s spletnim strežnikom je predpogoj za razumevanje dnevniških zapisov spletnega strežnika, saj je ta komunikacija

generator zapisov v dnevniku. Spletni brskalnik komunicira preko protokola *HTTP* (HyperText Transfer Protocol), za varen prenos pa se uporablja protokol *HTTPS*. Slika 2.11 prikazuje, kako komunikacija poteka po korakih [43]:

(1) Uporabnik klikne na povezavo (URL) na spletni strani, npr. `novice.si`. Ko HTTP zahteva pride do spletnega strežnika, ta vrne zahtevano stran (v našem primeru `stran.html`).

(2) Ko je dokument `stran.html` v celoti prenešen na odjemalčevo stran, spletni brskalnik dokument v celoti pregleda za morebitne *reference* na ostale dokumente, ki jih dokument `stran.html` vsebuje in jih je potrebno prenesti na odjemalca, preden bo postopek v celoti zaključen. Spletni brskalnik mora ostale komponente dokumenta prenesti kot ločene zahteve. Naj opomnimo, da je uporabnik kliknil samo na povezavo `stran.html`. Vse ostale akcije se izvedejo samodejno v okviru prenešenega dokumenta `stran.html`. Zaradi višje hitrosti večina brskalnikov prenose komponent na strani izvaja vzporedno (10 ali več zahtev naenkrat).

(3) Brskalnik najde povezavo na sliko – `logo.gif`, iz URL naslova se vidi, da se nahaja na istem strežniku (`novice.si`). Spletni odjemalec pošlje še eno zahtevo na strežnik, ta pa mu kot rezultat pošlje sliko.

(4) – oglaševanje: spletni brskalnik nadaljuje na naslednji referenci in najde zahtevo za prenos slike s spletnega mesta `oglas-pasica.si`. Pošlje se zahteva strežniku z oglasi `oglas-pasica.si`. Ta najprej vpraša brskalnik za piškotek, ki ga je morda prejšnjič poslal odjemalcu. Strežnik z oglasi dobi piškotek, ga obdela in na podlagi vsebine pošlje zahtevano sliko (oglasno pasico) brskalniku glede na vrednost piškotka. Strežnik z oglasi zabeleži kdaj, komu in katero pasico je poslal. Če brskalnik ne bi imel piškotka, bi mu oglasni strežnik poslal obstojni piškotek in naključno oglasno pasico.

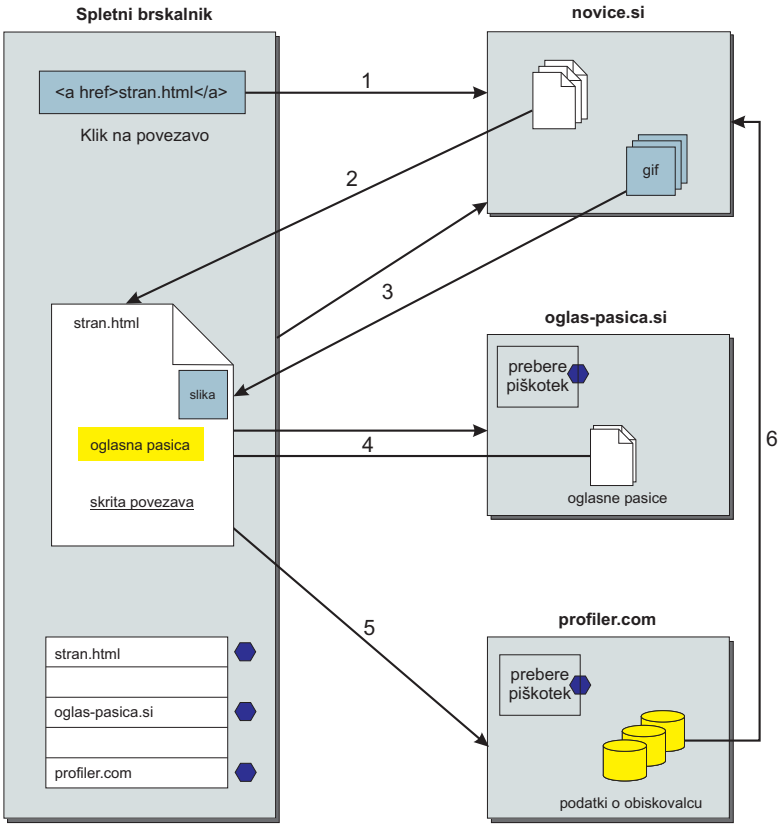
Vir zahteve: HTTP zahteva iz brskalnika strežniku `oglas-pasica.si` je nosila delček informacije, ki jo imenujemo *vir zahteve* ali *naslov vira zahteve* (ang. referrer). Vir zahteve je URL, ki je odgovoren za povezavo na spletni strani. V našem primeru je vir zahteve `novice.si/stran.html`. To ni spletni brskalnik, to je naslov URL s katerega je narejena zahteva HTTP za pasico na oglasnem strežniku. Strežnik z oglasi `oglas-pasica.si` tako dobi informacijo o tem, v okviru katerega spletnega mesta je bila pasica prikazana uporabniku.

Če si spletni mesti (strežnika) delita dnevniške datoteke in atribute strani, potem lahko izvemo tudi kakšen je bil namen spletne strani, na kateri se nahaja pasica in ne samo naslov URL.

(5) Medtem ko je stežnik za oglase zadolžen za dostavo pravilne vsebine na spletne strani, je *profiler* strežnik, katerega naloga je zbrati čimveč podatkov o

uporabniku za potrebe oglaševanja ali prilagajanja vsebine uporabniku. V našem dokumentu `stran.html` se nahaja *skrito polje*, ki vsebuje zahtevo za prenos posebnega dokumenta iz strežnika `profiler.com`. Ko zahteva pride do strežnika, ta najprej poskuša pridobiti svoj piškotek iz prejšnjih obiskov. Piškotek vsebuje ID, ki služi kot identifikacija uporabnika in šifra podatkov o uporabniku v podatkovni bazi profilerja. Profiler lahko podatke pošlje nazaj brskalniku (`stran.html`), ki bodo ob naslednji osvežitvi poslani na strežnik `novice.si`, ali pa po posebni poti nemudoma obvesti strežnik `novice.si`, da je uporabnik identificiran in aktiven **(6)**. Spletni strežnik `novice.si` bo lahko ustrezno prilagodil vsebino spletne strani poznanemu uporabniku.

Po vseh opisanih korakih, ki se zgodijo v zelo kratkem času, se z višanjem prenosnih hitrosti sorazmerno krajša, spletni brskalnik uporabniku prikaže zahtevano spletno stran. Šele s tem je zahteva za spletno stran (in s tem tudi komunikacija) dokončno zaključena. Iz opisanega primera vidimo, kako zahtevna komunikacija poteka v ozadju spletnega deskanja. Na sliki 2.11 so uporabljeni trije strežniki, funkcijo vseh treh lahko opravlja en sam strežnik.



Slika 2.11: Shematski prikaz komunikacije med odjemalcem in spletnim strežnikom. Postopek je razdeljen na šest korakov.

2.10.2 Dnevniška datoteka spletnega strežnika

Spletni strežniki so gotovo najbolj bogat in najpogosteje uporabljen vir podatkov o klikotoku. Oblika dnevniških datotek spletnih strežnikov je, ne glede na to, kateremu strežniku pripadajo, predpisana s standardom organizacije W3C. W3C predpisuje standardne elemente dnevnikov, ne pa tudi vsebine. Večina spletnih strežnikov uporablja kot privzeti format zapisa v dnevniško datoteko format CLF (Common Log Format), ki vključuje sedem podatkovnih elementov: IP odjemalec, ime odjemalca, uporabniško ime (če je uporabnik prijavljen), čas dostopa, ime datoteke in velikost datoteke. Več podatkov o dostopu najdemo pri W3C formatu ECLF (Extended Common Log Format), ki ima dodana še dva podatkovna elementa: vir zahteve (referrer) in vrsto odjemalca (user agent). Vir zahteve predstavlja spletno stran, iz katere je uporabnik zahteval trenutno obravnavano stran v dnevniku spletnega strežnika. Drugi element vsebuje podatek o imenu in različici spletnega brskalnika ter operacijskem sistemu na odjemalcu. S ponujenimi dvemi dodatnimi elementi ponuja format ECLF veliko večje možnosti v fazi obdelave dnevniških zapisov. Tabela 2.1 prikazuje primerjavo med formatoma CLF in ECLF. Prikazana je struktura dnevnika in opis elementov, ki jih najdemo v obeh formatih.

Tabela 2.1: Primerjava med formatoma spletnega dnevnika CLF in ECLF

Podatkovni element	**CLF**	**ECLF**	**Opis**
host (ime odjemalca)	•	•	Naslov IP ali polno ime domene (počasno).
ident (odjemalec)	•	•	Identiteta odjemalca, če je omogočena, sicer znak '-'.
authuser (uporabnik)	•	•	Če je spletni vir zaščiten z geslom uporabniški ID, ki je bil uporabljen, sicer prazen vnos.
time (datum in čas)	•	•	Čas zahteve na strežniku v obliki CLF [YYYY-MM-DD hh:mm:ss]
request (zahteva)	•	•	Polna pot do zahtevanega vira na strežniku.
status (HTTP status)	•	•	Statusna koda HTTP, vrnjena odjemalcu.
bytes (velikost)	•	•	Količina podatkov, ki se vrne odjemalcu (v bajtih), brez HTTP glave.
referrer (vir zahteve)		•	URL naslov dokumenta, ki je uporabnika usmeril k nam.
user-agent (brskalnik)		•	Ime in različica spletnega brskalnika + platforma.
filename			Ime datoteke.
cookie (piškotek)			Vrednost piškotka, ki ga je poslal spletni brskalnik iz lokalne datoteke za piškotke.
...			

Standarda CLF in ECLF nista dovolj za današnje potrebe pri zbiranju informacij, zato praktično vsi spletni strežniki omogočajo beleženje tudi drugih podat-

kovnih elementov (npr. piškotek, IP strežnika). Število elementov je omejeno s protokolom HTTP, ki v osnovi ne nudi veliko informacije [43]. Pri instalaciji in nastavitvi spletnega strežnika morajo razvijalci in skrbniki spletnega mesta določiti, kateri podatkovni elementi bodo potrebni za implementacijo vseh željenih funkcij. Od izbora podatkovnih elementov je odvisna tudi kakovost analiz podatkov klikotoka. Odločimo se lahko za celoten nabor podatkovnih elementov, ki so na voljo. To ima za posledico zelo velike in za manipuliranje zahtevne datoteke, hkrati pa obilo uporabnih podatkov. Druga možnost je uporaba samo nekaj osnovnih parametrov, ki prinesejo omejen nabor podatkov, vendar dovolj majhne in obvladljive dnevniške datoteke [3]. Odločitev o formatu zapisa v dnevniško datoteko spletnega strežnika je potrebno skrbno pretehtati.

2.10.3 Pomembnejši elementi dnevnika

Analiza spletnih dnevnikov je lahko zelo zapletena in nudi svojevrsten izziv. Kljub temu, da se za vsak zahtevek naredi zapis v dnevnik, se soočamo z različnimi težavami. Zapisi v dnevniku nimajo podatka o uporabniški seji. Vsaka vrstica v dnevniku torej predstavlja samostojen zahtevek brez povezave na ostale zahtevke v uporabniški seji. Ker strežnik laho streže več uporabnikom obenem, se zapisi različnih uporabniških sej v dnevniški datoteki se med sabo prepletajo. To seveda otežuje postopek združevanja vseh zahtev uporabnika v uporabniško sejo. Ta proces imenujemo *osejevanje* (ang. sessionization). Za osejevanje pomembnejši podatkovni elementi dnevniške datoteke so:

- **IP-naslov odjemalca**. IP-naslov je eden izmed najpomembnejših elementov dnevnika. Na podlagi IP-naslova lahko spletni strežnik s pomočjo protokola reverse address lookup ugotovi obiskovalčevo domeno, ki včasih razkrije državo ali organizacijo obiskovalca. Ta možnost se uporablja redko, ker predstavlja za strežnik 40% dodane obremenitve. Domenske podatke iz IP-naslova raje ugotavljamo v postopku poprocesiranja. Zaradi prevlade dinamičnih IP-naslovov nad statičnimi in dostopa izza požarnih zidov podjetij ter proxy strežnikov [71] (vsi računalniki v podjetju so navzven predstavljeni z istim IP-naslovom) ime domene včasih pove bore malo, ravo tako ni možna popolnoma pravilna identifikacija obiskovalcev s pomočjo IP-naslova.

- **Datum in čas** – time. V dnevnik se običajno zapiše čas, ko se je spletni strežnik končal ukvarjati z zahtevo. Čas se beleži s sekundno natančnostjo. Časovni žig je pomemben element v postopku osejevanja. Vsaka uporabniška seja se začne na nek dan ob nekem trenutku, in se ob nekem trenutku tudi konča (običajno istega dne). Na podlagi izteka časovne omejitve od zadnje zahteve (ang. timeout) sklepamo, da se je seja končala. Shahabi in sod. [86] ta čas imenujejo *čas ogleda strani*. Različni avtorji predlagajo različne vrednosti [13, 105], običajno pa se uporablja čas 30 min [20]. Če po izteku naletimo na novo zahtevo z istega IP-naslova sklepamo, da gre za novo uporabniško sejo.

Atributa datum in čas sta uporabna neposredno v podatkovnem skladišču in pri združevanju zapisov iz različnih datotek.

- **Dejanska zahteva spletnega brskalnika** – request. Primer tipične zahteve:

  ```
  GET /images/slika.gif HTTP/1.0
  ```

  Zahteva je sestavljena iz HTTP metode, (v našem primeru zahteva za nek vir), URI segmenta (uniform resource identifier) in različice protokola. POST metoda pošlje podatke strežniku. S stališča preiskovanja dnevnikov za potrebe prepoznavanja obnašanja uporabnikov sta zanimivi predvsem metodi GET in POST. Prva se pojavlja v skoraj 99% zahtev in pomeni, da je uporabnik kliknil na hiperpovezavo (ang. hyperlink) in s tem sprožil zahtevo za prenos spletne strani. Povedano drugače, uporabnik si je stran samo ogledal. Zanimiva je tudi metoda POST, ki predstavlja uporabnikovo akcijo pošiljanja vsebine na spletno stran [3]. To se zgodi, ko uporabnik izpolni podatke v obrazcu in ga pošlje na spletno mesto. Veliko takih akcij najdemo pri spletnih aplikacijah na vseh straneh, ki predstavljajo vnos podatkov v sistem. Po izvršitvi akcije, ki se nahaja za akcijskim elementom spletne strani, se spletna stran spremeni ali prikaže druga, uporabnikovi akciji ustrezna stran.

- **Status** – trimestna koda stanja, ki se vrne odjemalcu kot odgovor na zahtevo. Pove kako se je zahteva končala. Omogoča sledenje in analiziranje dogodkov na strežniku. Okvirni pomen pomembnejših skupin kod najdemo v tabeli 2.2.

- **Vir zahteve** – referrer. Predstavlja izvorno spletno stran (URL), preko katere je obiskovalec prišel na našo spletno stran. S pomočjo tega podatka spletna mesta lahko spremljajo promet glede na to, od kod vse prihajajo obiskovalci na spletno mesto. Uporabimo ga lahko v varnostne in statistične namene. Ker ta podatek lahko krši zasebnost, omogočajo nekateri brskalniki izključitev pošiljanja tega podatka. V zadnjem času se pojavljajo nekatere zlorabe vira zahteve, kot so: spreminjanje podatka z namenom nepooblaščenega dostopa do spletne strani (ang. referrer spoofing) in pošiljanje spremenjenega podatka o viru zahteve (ang. referrer spam).

  V zadnjem času so najpogostejši izvor zahtev znani spletni iskalniki, npr. Google, Msn, Najdi.si. Za spletna mesta so pomemben vir tudi druga spletna mesta, ki usmerijo obiskovalca nanje. Načini preusmeritve se pojavljajo v obliki hiperpovezav, grafičnih elementov, oglasnih pasic, in drugih elementov. V postopku obdelave lahko vire indentificiramo in ovrednostimo njihovo pomembnost [3].

- **Odjemalec** – user-agent. Vsebuje podatek o imenu in različici odjemalca, ki je ustvaril zahtevo in operacijski sistem pod katerim odjemalec teče. Spletni strežnik ta podatek lahko uporabi za ugotavljanje, katere funkcije so na tem brskalniku podprte in katere ne. Na podlagi tega podatka ponavadi zaznamo

in ločimo robotke spletnih iskalnikov. Ti namreč neprestano preiskujejo spletna mesta in generirajo zapise v dnevnikih. Za spletni dnevnik pomeni obisk robota povsem isto kot obisk običajnega uporabnika. Vrstica v dnevniku se razlikuje le v atributu `odejmalec`. Vsak robotek ima namreč svoj podpis. Tabela 2.3 prikazuje podpise nekaj znanih robotov. Primer podpisa odjemalca Internet Explorer, ki teče na operacijskem sistemu Windows 2003:

```
Mozilla/4.0+(compatible;
+MSIE+6.0;+Windows+NT+5.2;+SV1;+.NET+CLR+1.1.4322)
```

- **Piškotek** – cookie. Protokol HTTP ne omogoča ugotavljanja stanj in prehodov med njimi (ang. stateless). Ta lastnost lahko pomeni težave pri enolični identifikaciji uporabniške seje. Kot odgovor na to pomanjkljivost je bil vpeljan mehanizem *izmenjave piškotkov* (cookie). Mehanizem je bil med ponudniki spletnih strežnikov in odjemalcev dobro sprejet in je dandanes vgrajen v arhitekturo spletnih strežnikov kljub temu, da uradno ni del standarda HTTP 1.1. Mehanizem piškotkov omogoča strežniku, da pošlje niz znakov odjemalcu z namenom, da ga v prihodnje lahko prebere. Piškotek se hrani v lokalni bazi odjemalca.

  Običajno piškotek hrani informacijo o enolični identifikaciji, ki jo izdela spletni strežnik. Ko uporabnik naslednjič obišče spletno mesto, se ta podatek lahko pošlje nazaj spletnemu strežniku, ki lahko identificira obiskovalca. Piškotki lahko hranijo tudi druge vrste informacije, kot so obiskane strani, kupljeni izdelki, itd. Piškotek ne sme biti večji od 4 Kbajtov, torej lahko zapišemo v piškotek samo omejeno količino podatkov [71]. Piškotek je lahko *trajen* (ang. persistent) ali *začasen* (ang. session level). Trajni piškotek se zapiše na disk v brskalnikovo shrambo za piškotke in velja do datuma, ki ga določi strežnik. Zaradi varovanja zasebnosti lahko posamezne piškotke pišejo, berejo in spreminjajo le HTTP odjemalci iz domene, ki je shranjena v datoteki piškotka. Začasni piškotek se nahaja samo v delovnem spominu odjemalca in velja le toliko časa, dokler traja uporabniška seja. Izgine takrat, ko zapremo odjemalca, ki komunicira s spletnim strežnikom. Vsebina piškotka je preprosta. Stežnik lahko nastavlja naslednje vrednosti piškotka:

  - name - ime piškotka,
  - **value** - vrednost piškotka,
  - expires - časovna oznaka poteka veljavnosti piškotka (GMT),
  - domain - ime domene spletnih strežnikov, ki lahko berejo piškotek,
  - path - pot v domeni, kjer piškotek velja,
  - secure - ali mora biti piškotek poslan preko varne povezave (HTTPS).

  Piškotki seveda niso brez pomanjkljivosti. Prva težava je, da lahko istemu uporabniku dodelimo več piškotkov, če se povezuje z različnih računalnikov

ali nasprotno, več uporabnikov lahko uporablja isti računalnik in s tem iste piškotke. V skladu z varovanjem osebnih podatkov lahko uporabnik v nastavitvah brskalnika onemogoči uporabo piškotkov. Tudi če brskalnik sprejema piškotke, jih lahko uporabnik nekaj izbirno pobriše. Omejeno je tudi število piškotkov. Samo 20 piškotkov je dovoljeno hraniti za določeno domeno, brskalnik pa lahko hrani največ 300 piškotkov [71].

Tabela 2.2: Statusne kode protokola HTTP.

Koda	**Skupina dogodkov**	**Opis skupine**
2xx	Uspeh	Strežnik je uspešno obdelal zahtevo.
3xx	Preusmeritev zahteve	Odjemalec mora izvesti dodatna dejanja za dokončanje zahtevka. Dejanje je lahko opravljeno z ali brez interakcije z uporabnikom.
4xx	Slab zahtevek	Prišlo je do napake, ki je najverjetnejše posledica napake v sami zahtevi. Strežnik ni mogel izvršiti zahteve. Najbolj znana koda iz te skupine je 404 – spletni vir ni bil najden.
5xx	Strežniška napaka	Pri obdelavi zahteve je prišlo do napake na strežniku. Napaka ni posledica zahteve same ampak notranjega stanja strežnika (npr. 500 – napaka v programski kodi, ki izdela dokument za odjemalca).

Tabela 2.3: Pogostejši spletni iskalniki in običajne ključne besede v podpisu.

Iskalnik	Naslov iskalnika	Ključne besede v podpisu
Google	www.google.com	Googlebot
Najdi.si	www.najdi.si	Najdi.si/3.1
Yahoo	www.yahoo.com	Yahoo!+Slurp
Microsoft MSN	www.msn.com, www.bing.com	msnbot
Gigablast	www.gigablast.com	Gigabot
Alta vista	www.altavista.com	Scooter/3.3

2.10.4 Mehanizem za procesiranje klikotoka

Uvod

Ko smo ugotovili, kateri podatki o klikotoku so nam na voljo iz dnevnikov spletnih strežnikov, moramo najti pot kako te podatke shraniti v skladišče spletnih podatkov.

V veliko primerih lahko za procesiranje podatkov o klikotoku uporabimo standardne ETL procese, ki smo jih uporabili pri drugih podatkovnih skladiščih. Kljub temu pa obstaja nekaj pomembnih razlik, ki se jih moramo dobro zavedati. Razviti moramo model za sistem ETL, ki ga imenujemo *mehanizem za poprocesiranje podatkov o klikotoku*.

2.11 Področna podatkovna skladišča

2.11.1 Uvod

V nadaljevanju bomo predstavili logični načrt področnega podatkovnega skladišča za potrebe spletnega študijskega IS [75] in spletne trgovine. Logični načrt je predstavljen v obliki dimenzijskega modela. Uporabili bomo dimenzijski podatkovni model, ker je najbolj primeren format za predstavitev podatkov v podatkovnem skladišču [45]. Dimenzijski model je alternativa tradicionalnemu entitetno-relacijskemu (E-R) modelu, ki se bolj splošno uporablja v bazah podatkov.

Zvezdna shema je tehnika, ki se uporablja pri večdimenzionalnem modeliranju. Podatkovno skladišče je sestavljeno iz ene ali več zvezdnih shem. Zvezdna shema je sestavljena iz centralne tabele dejstev in več dimenzijskih tabel, ki vsebujejo denormaliziran opis shranjenega dejstva. Center zvezdne sheme predstavlja tabela dejstev, krake zvezde pa predstavljajo dimenzijske tabele. V tabeli dejstev se nahajajo ključi dimenzij in merljiva dejstva. Merljiva dejstva so številska. Lahko so seštevna (ang. aditive) ali polseštevna (ang. semi-aditive) glede na čas [43, 45].

Med dimenzijskim in (E-R) modeliranjem obstaja bistvena razlika predvsem v uporabi. Entitetno relacijski model se uporablja v transakcijskih sistemih. Predvsem je njegov namen odpravljanja redundance v podatkih - podatek naj bi bil zapisan v podatkovni bazi samo enkrat in na enem mestu. Primeren je za obdelavo velike količine trancakcij, saj v normaliziranem modelu vsaka transakcija praviloma vpliva le na minimalno število tabel in zapisov. Taka zasnova podatkovne baze pa ni primerna za analitsko okolje. Veliki podatkovni modeli so namreč zaradi svoje širine in zapletene zgradbe precej nepregledni. Poleg tega pa zahtevnejše poizvedbe, ki vključujejo več deset tabel s seštevanji in preštevanji drastično zmanjšajo odzivnost baze podatkov. Takšni sistemi so za uporabnika, ki sprejema kritične poslovne odločitve, neprimerni.

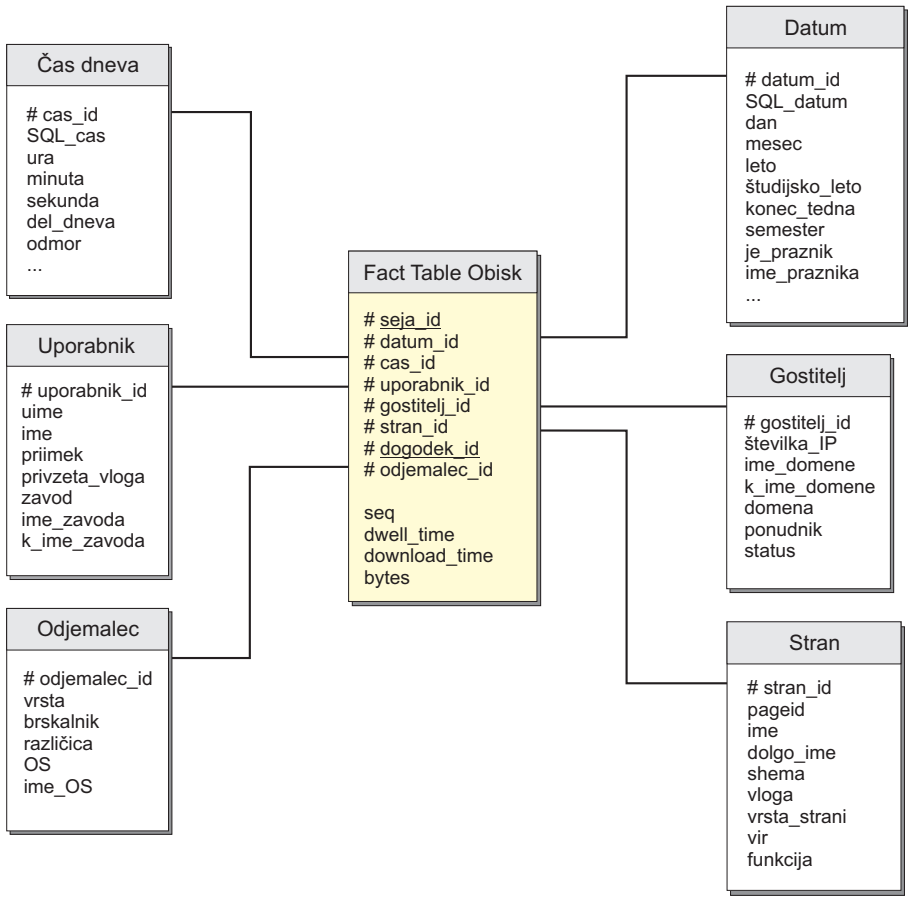
Dimenzijsko modeliranje je bolj usmerjeno k uporabniku, preglednejše in lažje obvladljivo. Tudi zahtevne in komplicirane poizvedbe so učinkovite in dajejo odgovore pravočasno. Takšni modeli so torej primerni za izvajanje analiz, poslovno poročanje, odkrivanje zakonitosti v podatkih in napovedovanje trendov. Za transakcijske sisteme niso primerni zaradi podvajanja podatkov, različne zrnatosti podatkov in hitrosti izvajanja.

2.11.2 Področno podatkovno skladišče za študijski IS

Logični načrt spletnega področnega podatkovnega skladišča predstavlja zvezdna shema, ki je sestavljena iz tabele dejstev na sredini in osmih dimenzij: seja, datum, čas, uporabnik, stran, gostitelj (ang. host), dogodek in odjemalec (ang. user agent). Dimenziji seja in dogodek sta *izrojeni* (razdelek 2.7.2). Degenerirana dimenzija nima pripadajoče dimenzijske tabele ampak ima samo ključ v tabeli dejstev. Zvezdno shemo prikazuje slika 2.12.

Tabela dejstev

Za zrno tabele dejstev smo izbrali posamezen dogodek zahteve neke spletne strani. *Zrno tabele dejstev* predstavlja nivo najbolj natančnega podatka, ki ga lahko shranjujemo v sistemu. Pri izbiri zrna smo upoštevali načelo, da izberemo tako drobno zrno, kot je mogoče. Ker vse možnosti uporabe niso natančno definirane, smo s tem omogočili najširši možen spekter analiz. Tabela dejstev vsebuje sestavljen ključ,



Slika 2.12: Zvezdna shema za spletno podatkovno skladišče študijskega IS.

ki je sestavljen iz ključev dimenzijskih tabel in štirih merljivih dejstev. Vsak za-

pis v tabeli dejstev ustreza eni zahtevi za stran na aplikacijskem strežniku v neki uporabniški seji (zrno tabele dejstev). Za vsako zahtevo strani, se hranijo naslednja merljiva dejstva [75]:

- dwell_time - število sekund, kolikor je imel uporabnik stran prikazano na zaslonu (čas branja strani).
- bytes_transferred - količina prenešenih podatkov s strežnika v bajtih. Sem se šteje velikost same strani in vseh elementov na strani.
- download_time - čas nalaganja strani na uporabnikov zaslon (v sekundah).
- page_sequence - zaporedna številka strani v celotni sekvenci obiskanih strani v seji.

Prva tri merljiva dejstva so seštevna in jih lahko seštevamo in agregiramo po vseh dimenzijah. Nad tabelo dejstev lahko izvajamo tudi štetje zapisov in na ta način najdemo odgovor na marsikatero zanimivo vprašanje.

Opis dimenzij

Dimenzija datum je ena izmed dveh časovnih dimenzij v zvezdni shemi. Časovno dimenzijo smo razdelili na dva dela zaradi izboljšanja performans sistema in v izogib nepotrebnemu podvajanju podatkov. Dostop do strani v tabeli dejstev merimo na sekundo natančno. Če bi hoteli samo eno časovno dimenzijsko tabelo, bi potrebovali 86.400 zapisov na dan, kar pomeni na leto 31.536.000 zapisov. Z delitvijo dimenzije v dva logična zaključena dela, smo se izognili takemu številu zapisov. Dimenzija datum vsebuje posebne atribute, ki so pomembni za analiziranje podatkov po datumih, na primer: počitnice, semester, študijsko leto, ipd. Za vsako leto ima 365 zapisov, kar jo uvršča med male dimenzije.

Dimenzija *čas dneva* ima en zapis za vsako sekundo v dnevu, kar pomeni, da ima 86.400 zapisov. Vsebuje atribute, ki opisujejo čas znotraj dneva: ura, minuta, sekunda, del dneva, čaz malice, čas kosila, število minut čez polnoč, ipd. Postopek polnjenja za dimenzijo se izvede samo enkrat, nadaljnje osveževanje dimenzije ni potrebno. Glede na število zapisov jo še vedno uvrščamo med male dimenzije.

Dimenzija *uporabnik* vsebuje podatke o uporabnikih študijskega, spletnega IS. Vsaka zapis pripada enemu uporabniku sistema in vsebuje vse njegove podatke. Uporabnik lahko nastopa v eni izmed štirih vlog: študent, učitelj, administrator ali skrbnik sistema. Uporabniki se morajo prijaviti v sistem, da lahko uporabljajo sistem. Zaradi tega imamo v spletnem podatkovnem skladišču opravka predvsem z znanimi uporabniki. To odpravlja tudi problem identifikacije anonimnih spletnih uporabnikov, ki je tipičen in netrivialen izziv pri analizi dnevniških zapisov klikotoka. Hranijo se tudi zapisi anonimnih uporabnikov, t.j. uporabnikov, ki se ne prijavijo v sistem. Atributi dimenzije uporabnik so razdeljeni na dva dela: atributi, ki so definirani za vse uporabnike (šifra uporabnika, uporabniško ime, vrsta) in atribute za uporabnike, ki so se prijavili v sistem (ime, priimek, privzeta vloga).

Dimenzija *stran* opisuje strani študijskega, spletnega IS. Vsak zapis v dimenziji pripada eni strani aplikacije. Vsaka stran iam enolični identifikator, ki omogoča popoln opis strani in definicijo atributov strani. Tipični atributi strani so: šifra strani, polno ime, kratko ime, vrsta strani (sttična, dinamična stran), del aplikacije, ki ji pripada, skupina strani, ipd. Ta dimenzija omogoča generiranje poročil o uporabi funkcij sistema.

Dimenzija *gostitelj* (ang. host dimension) hrani podatke o spletnih lokacijah uporabniških računalnikov (IP naslov, celotno ime domene, kratko ime domene, ponudnik storitev). Do določene mere natančno lahko ugotovimo tudi državo v kateri se nahaja IP naslov. Atribute za določen zapis v dimenziji lahko definiramo samo takrat, ko za IP naslov obstaja vnos v hierarhiji imenskih strežnikov.

Dimenzija *dogodek* pove, kakšne vrste dogodek se je zgodil nad določeno stranjo v točno določenem trenutku. V grobem ločimo med dvema vrstama dogodkov: prva zahteva strani oz. funkcije sistema in osvežitev strani sistema. Ker razločuje samo med dvemi vrstami dogodkov, smo jo zgradili kot degenerirano dimenzijo.

Dimenzija *odjemalec* opisuje odjemalca, s katerim je uporabnik dostopal do aplikacijkega strežnika. Posredno lahko pridemo tudi do podatka o operacijskem sistemu in nekaterih drugih osnovnih podatkih o uporabnikovem računalniku. Atributi te dimenzije so: vrsta odjemalca, ime brskalnika, različica brskalnika, platforma, operacijski sistem, ipd.

Dimenzija *seja* omogoča združevanje več zaporednih zahtev strani v skupino, ki jo imenujemo seja. Seja vsebuje vse zahteve za strani, ki jih je uporabnik naredil znotraj določenega časovnega intervala. To je degenerirana dimenzija, ker smo vse njene atribute prenesli na ostale dimenzije. To dimenzijo bi lahko implementirali tudi kot dejansko dimenzijo z atributi: število obiskanih strani v seji, dolžina trajanje seje, število vseh različniz zahtev za stran, ipd. Te podatke lahko dobimo tudi iz obstoječe zvezdne sheme, ob pogosti uporabi pa bi bilo smiselno razmisliti o predizračunu v fazi predprocesiranja.

2.11.3 Področno podatkovno skladišče za spletno trgovino

Zvezdna shema področnega podatkovnega skladišča je zelo podobna zvezdni shemi na sliki 2.12. Razlika je le v dimenziji uporabnik, ki je ima drugačne atribute. Pri spletnem študijskem IS imamo opravka z znanimi, uporabniki, ker se je za uporabo potrebno prijaviti. Pri spletni trgovini pa večinoma ne vemo kdo so uporabniki, kakšna je njihovo ime in ostali podatki. Tudi, če se uporabnik prijavi, je o njem na voljo manj podatkov.

Atributi dimenzije uporabnik za spletno trgovino so: uporabnik_id, user, piškotek, robot, username, ime in priimek. Med temi izpostavimo atribut *robot*, ki pove, ali je vsebino generiral spletni pajek ali uporabnik za računalnikom.

2.12 Implementacija postopkov polnjenja

2.12.1 Izbor orodij

Ko smo imeli potanko opredeljen logičen načrt spletnega podatkovnega skladišča smo izbrali primerno orodje in načrt implementirali. Pri izboru orodja za implementacijo smo se oprli na pretekle izkušnje pri gradnji podatkovnih skladišč. Za implementacijo spletnega podatkovnega skladišča smo uporabili orodje Microsoft SQL Server 2008, ki omogoča učinkovito izgradnjo in vzdrževanje podatkovnih skladišč vseh velikosti. Združuje popolno relacijsko in večdimenzionalno podatkovno bazo in omogoča podporo za različne načine uporabe, od e-poslovanja, do podpore poslovnim procesom in seveda podatkovnega skladiščenja. Arhitektura za učinkovito izgradnjo podatkovnega skladišča mora vsebovati podporo za:

- Izdelavo izvlečkov iz podatkovnih virov, preoblikovanje in nalaganje podatkov v podatkovno skladišče.
- Oblikovanje strukture podatkov v denormalizirano obliko zvezdne sheme ali v tretjo normalno obliko.
- Prenašanje podatkov v področna podatkovna skladišča kjer so ponavadi uporablja večdimenzionalni sistem za upravljanje podatkov (npr. OLAP strežnik).
- Izdelave poročil v najširšem smislu, kjer vir podatkov zanje predstavlja podatkovno skladišče in/ali področna podatkovna skladišča. Poročila imajo lahko obliko tiskanih ali poglobljenih večdimenzionalnih poročil.

SQL Server 2008 zagotavlja orodja za uspešno izdelavo vseh naštetih nalog. SQL Server Integration Services (SSIS) omogoča izdelavo in vzdrževanje postopkov za čiščenje in preoblikovanje podatkov (ETL). SSIS je funkcijsko bogato orodje. Največ se uporablja pri sistemih, kjer se podatki neprestano preoblikujejo iz ene v drugo obliko. Orodje SSIS ni omejeno samo na uporabo pri gradnji podatkovnih skladišč, vsebuje pa vrsto komponent, ki gradnjo podatkovnih skladišč zelo olajšajo. V okviru dostopa do podatkov je poenostavljen proces izdelave izvlečkov. Tabele, ki tvorijo podatkovno skladišče, se nahajajo v okviru relacijske podatkovne baze.

OLAP strežnik SQL Server Analysis services (SSAS) podatke predstavi v večdimenzionalni obliki, prilagojeni za hitro izdelavo poročil in enostavno razumevanje podatkov. SSAS je integrirana in razširljiva množica komponent, ki omogoča gradnjo večdimenzionalnih podatkovnih baz in njihovih podatkovnih struktur (kock). Strežnik SSAS je zelo fleksibilen in omogoča več različnih načinov shranjevanja podatkov in agregatov. Na voljo je veliko čarovnikov za enostavno izdelavo dimenzij in kock. Vsebuje tudi industrijsko podprte standarde za podporo odkrivanju zakonitosti v podatkih (ang. data mining). Za podporo poročanju SQL Server nudi Reporting Services (SSRS), ki omogoča izdelavo različnih analiz vključno z odlično povezavo z orodji Excel in Word.

Kot SUPB za spletno podatkovno skladišče nam je služila relacijska podatkovna baza SQL Server. Postopek za čiščenje in preoblikovanje podatkov (postopek ETL) smo implementirali s pomočjo orodja SSIS in lastnih programov, napisanih v jeziku C#, ki smo jih po potrebi klicali iz orodja SSIS. Postopek ETL je najbolj zapleten in časovno potraten del podatkovnega skladiščenja zato smo, kjer je bilo možno, uporabili orodje SSIS. SSIS je načrtovan z mislijo poenostaviti proces razvoja postopka ETL. Grafično ogrodje omogoča enostaven dostop do virov, določanje predhodno definiranih operacij in drastično zmanjšanje števila programskih vrstic v primerjavi z višjenivoskimi ali skriptnimi jeziki. Orodje že v samem jedru podpira nekatere zapletenejše naloge, kot so npr. počasi spreminjajoče se dimenzije.

SSIS je sestavljen iz logičnih zaključenih celot, imenovanih paketi. Paket je logična zaključena celota, ki jo v fazi razvoja postopka gradimo in v fazi uporabe izvajamo. Zgrajen paket vsebuje povezave na vire podatkov različnih SUPB, naloge nad podatki in podatkovni tok, ki naloge povezuje med seboj. Pakete med sabo lahko poljubno gnezdimo. To nam omogoča razbitje postopka ETL na več manjših in obvladljivejših delov.

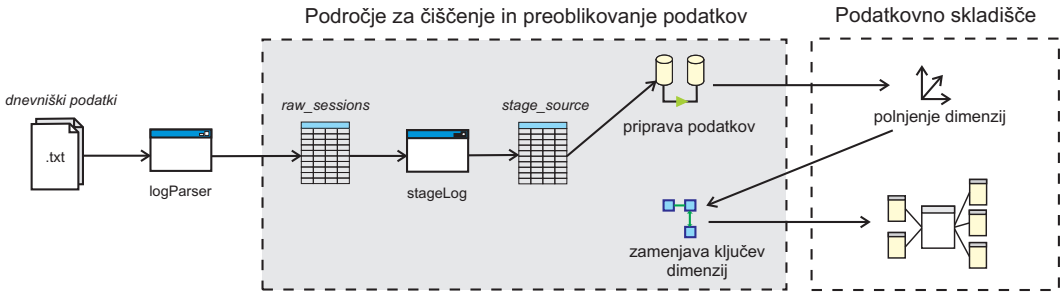
2.12.2 Izvedba postopka ETL

Postopek polnjenja podatkovnega skladišča je razdeljen na več delov. Večji del postopkov smo implementirali s pomočjo orodja SSIS, za bolj zapletene postopke pa smo izdelali aplikacije v jeziku C#. Tak primer je postopek za čiščenje in osejevanje podatkov o klikotoku, kjer je potrebno izdelati učinkovito rešitev, tako s stališča hitrosti kot upravljanja s pomnilnikom. Vsi postopki, vključno s klicem aplikacij, se izvedejo v okviru okolja SSIS. Postopek polnjenja je prikazan na sliki 2.13. Polnjenje podatkovnega skladišča je razdeljeno na tri večje dele:

- zajem podatkov o klikotoku, čiščenje in osejevanje,
- preoblikovanje in preverjanje podatkov v področju ETL,
- zapis podatkov v podatkovno skladišče.

Zajem podatkov

Podatke o klikotoku pridobimo iz dnevnika spletnega strežnika. Podatki so zapisani v obliki formatirane tekstovne datoteke. Struktura in pomen podatkov je podrobneje opisana v poglavju o opisu testnih podatkov. Postopek zajema podatkov, čiščenja, ugotavljanja uporabnikov in osejevanja smo izvedli s pomočjo programa `parseLog`, ki smo ga napisali v jeziku C#. Po izvedbi tega postopka smo v podatkovni bazi v tabeli `raw_sessions` dobili obdelan klikotok z dodanimi podatki o uporabniški seji (identifikacijska številka seje, čas trajanja seje, količina prenešenih podatkov). Klikotok še vedno vsebuje določena podvajanja, različna poimenovanja za isti vir, seje spletnih robotov še niso ločene od uporabniških. Za sistem e-Študent smo v fazi procesiranja s programom `parseLog` že identificirali prepletene seje.



Slika 2.13: Shematski prikaz postopka polnjenja spletnega podatkovnega skladišča.

Preoblikovanje podatkov

V drugem delu postopka ETL se podatki že nahajajo v podatkovni bazi v področju za čiščenje in preoblikovanje. Podatke prenašamo med tabelami in jih v s programi in rutinami spreminjamo ter dodajamo atribute. Program `stageLog` zapise prenese v tabelo `stage_source`. Pri tem številko sej prilagodi glede na obstoječe v podatkovnem skladišču, odstrani podvojene zapise, ki so posledica načina delovenja aplikacijske plasti, in loči seje spletnih robotov od uporabniških sej. Program `stageLog` identificira zahtevane strani v sistemu in zamenja z veljavno šifro v podatkovnem skladišču.

Preden podatke o klikotoku dodatno preoblikujemo in zapišemo v tabele zvezdne sheme, moramo pripraviti področje za čiščenje in preoblikovanje podatkov. Načeloma se postopki za pripravo pomožnih tabel izvedejo samo ob ob prvem polnjenju podatkovnega skladišča. Mednje sodi polnjenje časovne dimenzije, polnjenje dimenzije `dogodek` in polnjenje pomožne tabele za ugotavljanje pripadnosti IP številke določeni državi. Določene postopke pa je potrebno ponoviti v enakomernih časovnih razmakih. Preveriti moramo, če ima dimenzija `datum` zapise za časovno obdobje, za katero polnimo podatke v spletno podatkovno skladišče. Spletne aplikacije se spreminjajo, dodajajo se nove strani in uporabniki. V področje za čiščenje in preoblikovanje je potrebno prenesti podatke o vseh straneh aplikacije in uporabnikih. V nasprotnem primeru se lahko zgodi, da je v klikotoku zahtevana spletna stran za katero nimamo vseh potrebnih podatkov, ki jih hranimo v podatkovnem skladišču.

Preden zapišemo podatke o klikotoku v podatkovno skladišče, je potrebno osvežiti dimenzije. Obe časovni dimenziji smo že osvežili v okviru priprave področja za čiščenje in preoblikovanje. To lahko storimo zato, ker sta relativno neodvisni od podatkov. Za osvežitev ostalih dimenzij potrebujemo podatke o klikotoku. Na podlagi zapisov v tabeli `stage_source` moramo osvežiti dimenzije uporabnikov, strani in podatkov, povezanih z IP številko. V dimenzijske tabele skladišča je potrebno zapisati vse podatke, ki se nahajajo v klikotoku in jih še ni v trenutno veljavnih dimenzijskih tabelah. Primer: v dimenzijo uporabnik je potrebno zapisati vse nove uporabnike, ki pred tem še niso uporabljali spletne aplikacije. Podobno

moramo dodati vse od zadnjega polnjenja na novo dodane strani aplikacije. To velja tudi za dimenzijo `gostitelj`. Za vse nove IP številke je potrebno ugotoviti morebitno obstoječe ime domene, končnico in državo pripadnosti. V ta namen smo razvili aplikacijo `resolveDNS`. Zaradi počasnosti postopka se proces poizvedovanja domenskih strežnikov izvede po končanem glavnem polnjenju.

Zapis podatkov v skladišče

Po osvežitvi vseh dimenzijskih tabel, lahko pripravimo podatke o klikotoku za zapis v tabelo dejstev. V tabeli dejstev se nahajajo samo ključi dimenzij in merljiva dejstva. V tabeli `stage_source` je potrebno za vsak podatek, ki predstavlja dimenzijo, zamenjati vrednost v dimnezijski tabeli z njegovim ključem. Poskrbeti je potrebno, da se sestavljeni ključi zapisa ne podvajajo. Predvsem pa je potrebno paziti, da se številka seje ne podvaja s kakšno izmed obstoječih sej v tabeli dejstev. Po končani zamenjavi vrednosti s ključi pred zapisom v tabelo dejstev še enkrat preverimo integritene omejitve, da sredi postopka polnjeja tabele dejstev ne pride do napake. Zadnje dejanje je polnjenje tabele dejstev v enem kosu (ang. bulk load).

Zapisi, pri katerih pride do napake kjerkoli v postopku ETL, se preusmerijo v posebno tabelo `stg_napaka`. Tabela predstavlja dnevnik za učinkovito lociranje in odpravo napak. Preslikave in postopki vanjo zapisujejo napake, ki so posledica nepravilnosti podatkov in kršenih poslovnih pravil, preden se prekine izvajanje postopka ETL. Po končanem postopku polnjenja nam nudi vpogled v napake in razloge, ki so pripeljali do napak. Vsebina tabele se načelno briše ob vsakem ponovnem začetku izvajanju postopka ETL.

POGLAVJE 3

Analiza problema in možnih rešitev

3.1 Opis problema

3.1.1 Izzivi razpletanja sej

Uporaba in pomen spleta stalno naraščata, še posebej v poslovnem svetu splet postaja ključen faktor za uspeh podjetja. Vloga spletnih analitikov, ki se ukvarjajo z merjenjem in z ugotavljanjem zakonitosti v podatkih, ki pri tem nastajajo, je v zadnjih letih strmo pridobivala na pomenu.

Raziskovalno področje odkrivanja zakonitosti v spletnih podatkih se danes deli na tri področja: iskanje vzorcev v vsebini spletnih strani (ang. web content mining), iskanje vzorcev obnašanja uporabnikov (ang. web usage mining) in iskanje vzorcev v zgradbi spletnega mesta (ang. web structure mining). Za nas bo najbolj zanimivo drugo področje odkrivanja zakonitosti v podatkih. Podatke o obnašanju uporabnikov pri sprehajanju po straneh spletnega mesta dobimo iz klikotoka. Ne glede na vrsto spletnega mesta je potrebno pred izračunom za nas zanimivih podatkov izvesti posreden korak predpriprave podatkov (slika 3.1). V tem koraku identificiramo uporabnike, klikotoku dodamo podatek o seji in določimo pravilen vrstni red spletnih strani znotraj sej (postopek osejevanja). Sledi lahko še dodaten korak obdelave osejenih podatkov. Običajno se pri osejevanju zanašamo na osnovno predpostavko,

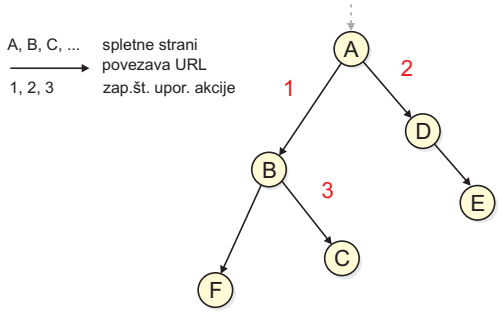
Slika 3.1: Koraki pri odkrivanju zakonitosti v spletnih podatkih (ang. web mining steps).

da časovni vrstni red klikov (v seji) opisuje pot spletnega uporabnika pri brskanju po spletnem mestu. Povedano drugače, predpostavljamo, da se uporabnik sprehaja po določenem spletnem mestu vedno samo z enim spletnim brskalnikom, v katerem imamo odprto in aktivno samo eno okno. Ta predpostavka pa v zadnjem času ne drži več nujno predvsem zaradi dveh razlogov. Na trgu so se pojavili spletni brskalniki s podporo zavihkom (ang. tabs) znotraj enega okna brskalnika, drugi razlog pa tiči v spremenjenih vzorcih obnašanja uporabnikov.

Možnost uporabe zavihkov znotraj spletnega brskalnika je spremenila način deskanja uporabnikov. Uporaba zavihkov je uporabnikom poznan koncept iz okenskih programov in spodbuja razbitje krovne naloge v več manjših, preprostejših podnalog. Mehanizem zavihkov v brskalniku uporabnike implicitno spodbuja, da daljši sphehod po določenem spletnem mestu razbijejo na poddele in ga opravijo v več zavihkih. Uporabnik se na spletnem mestu z uporabo zavihkov tudi lažje znajde. Na določenem mestu na poti po spletnem mestu odpre stran v novem zavihku, opravi določeno podnalogo v novem zavihku, potem pa se vrne na prvotno okno. Pogost primer: spletni kupec zaradi prehoda na digitalni način oddajanja kupuje nov TV aparat. Najde primeren izdelek, vendar ga zanimajo še potencialno cenejši izdelki. Odpre nov zavihek in v njem najde primerljivo ponudbo. Preklaplja med zavihkoma in primerja lastnosti, nato pa se odloči, da bo v prvem zavihku poskušal najti še boljšo ponudbo. Namesto zavihkov lahko uporabnik uporabi več oken brskalnika.

Ne glede na mehanizem zavihkov uporabniki velikokrat potrebujejo odprtih več oken brskalnika za isto spletno mesto obenem. To velja še posebej za napredne uporabnike pri uporabi spletnih aplikacij. Pri uporabi spletne aplikacije uporabnik vedno nastopa v določeni uporabniški vlogi. Naprednejši uporabniki pa imajo lahko več zadolžitev in opravljajo istočasno več nalog v več uporabniških vlogah. Lep primer je na skrbnik sistema, ki lahko nastopa tudi kot običajen uporabnik. Skrajno nerodno bi bilo, če bi se uporabnik po dokončanju naloge v prvi vlogi odjavil, prijavil v novi vlogi, opravil novo nalogo, odjavil in spet prijavil v prejšnji vlogi. Praviloma imajo torej takšni uporabniki odprtih več oken brskalnika in so v njih prijavljeni v različnih uporabniških vlogah. Po potrebi preklapljajo med okni in v okviru časovih rezin opravljajo delne ali zaključene naloge.

Zaradi enostavnosti bomo poimenovali uporabnikovo obnašanje, kjer je pri brskanju po splenem mestu uporabljeno eno samo okno, *linearno obnašanje* uporabnika. Obnašanje uporabnika, ki uporablja več oken ali več zavihkov za isto spletno mesto, pa bomo poimenovali *vzporedno obnašanje* uporabnika. Na podlagi vzporednega obnašanja uporabnikov v dnevniku spletnega strežnika nastanejo prepletene seje. V klikotoku imamo zabeležene obiske v zaporedju A, B, D, C. Razliko med linearnim in vzporednim obnašanjem lahko prikažemo s primerom poenostavljenega spletnega mesta, katerega načrt prikazuje slika 3.2. Vozlišča predstavljajo spletne strani, povezave med vozlišči pa prehode med stranmi. Številka ob povezavi pomeni zaporedno številko uporabnikove akcije. Uporabnik je torej najprej obiskal stran A, stran B, nato stran D na koncu pa še stran C. Tabela 3.1 prikazuje rekon-



Slika 3.2: Načrt preproste spletne strani.

strukcijo poti uporabnika na podlagi obiskanih strani. V prvem primeru predpostavljamo linearno obnašanje uporabnika, v drugem primeru pa vzporedno obnašanje uporabnika. Če pogledamo načrt spletne strani in predpostavimo linearno obnašanje uporabnika, vidimo, da je prehod med stranema D in C malo verjeten, ker strani nista neposredno povezani s hiperpovezavo. Ob predpostavki linearnega obnašanja bodo rekonstruirana seja s_n in časi ogledov strani zavajajoči. Ob predpostavki vzporednega obnašanja uporabnika dobimo pravilno rekonstruirani vzporedni seji uporabnika s_p.

Tabela 3.1: Rekonstrukcija sej na podlagi akcij uporabnika za spletno mesto na sliki 3.2. Indeksi predstavljajo številko seje.

Seja	Prehodi med stranmi
neprepletena: s_n	$A \rightarrow B, B \rightarrow D, D \rightarrow C$
prepletena: s_p	$[A \rightarrow B]_1, [A \rightarrow D]_2, [B \rightarrow C]_1$

3.1.2 Razpletanje sej

Za razumevanje obnašanja uporabnikov je potrebna pravilna rekonstrukcija sej. Nepravilna rekonstrukcija sej kvarno vpliva na kakovost analiz, ki kot vir uporabljajo osejene podatke o klikotoku uporabnikov. Berendt in sod. [9] navajajo, da na kakovost podatkov o obnašanju spletnih uporabnikov vplivajo težave z (i) razlikovanjem med zahtevami različnih uporabnikov (ii) določitvijo konca uporabniške seje in (iii) rekonstrukcijo vseh aktivnosti na spletnem mestu. Prepletene seje spadajo v tretjo skupino težav. Prepleteno sejo si lahko predstavljamo kot sejo, ki je sestavljena iz elementov sej dveh ali več različnih uporabnikov. Popolnoma jasno je, da takšne seje zakrivajo prvotni namen uporabnika, ki bi ga lahko izluščili iz ločenih sej. Iz prepletenih sej bomo torej težje izluščili pravilne vzorce. Takšne seje so daljše, verjetnost prehoda med dvema zaporednima stranema s_t in s_{t+1} v seji je lahko zelo majhna.

3.2 Pregled dosedanjega dela na področju klikotoka

3.2.1 Priprava podatkov o klikotoku

S predprocesiranjem podatkov o klikotoku se je ukvarjalo že veliko avtorjev. Vsi so prišli do ugotovitve, da je kakovostno predprocesiranje podatkov ključno za nadaljne analize.

Kohavi [47] se je v svojem članku ukvarjal z izzivi pri analizi podatkov o klikotoku. Navedel je vse pomembne predpostavke za kakovostno izvedbo iskanja zakonitosti v spletnih podatkih. Podatki o klikotoku so po njegovem mnenju izjemno primerna domena, vendar se je potrebno soočiti z izzivi pri predpripravi podatkov. Podrobno je opisal vse težave, ki jih imamo pri uporabi klikotoka iz dnevnikov spletnega strežnika. Kot alternativo dnevnikom spletnega strežnika je navedel uporabo aplikacijske plasti, tudi alternativo dnevnikom spletnega strežnika, in primerjal uporabo obeh pristopov. Posvetil se je tudi nekaterim odprtim problemom na področju klikotoka.

Veliko del v preteklosti se je osredotočalo na problem nepopolnih podatkov v dnevniku spletnega strežnika. Predpomnjenje namreč lahko povzroči nepopolne dnevnike spletnega strežnika. Ena izmed rešitev je pridobitev podatkov o klikotoku s strani odjemalca. Shahabi [85] je skoraj vse podatke o klikotoku pridobil iz spletnega brskalnika. Fenstermacher and Ginsburg [33] sta zajem interakcije med uporabnikom in odjemalcem razširila še na nekatere druge pisarniške aplikacije. Trdita, da na ta način dobimo veliko bolj popolno sliko uporabnikovega obnašanja na spletu. Catledge in Pitkow [13] sta za zajem klikotoka uporabila poseben vtičnik (ang. plugin) brskalnika, ki je hranil vse uporabnikove akcije v brskalniku. Težava takšnega pristopa je varnost in potrebno sodelovanje uporabnika.

3.2.2 Hevristike rekunstrukcij sej

Cooley in sod. [20] so se ukvarjali s tehnikami priprave podatkov o klikotoku za spletne analize. Predstavili so metode za identifikacijo uporabniških sej. Predprocesiranje so razbili v štiri korake: čiščenja dnevnika spletnega strežnika, identifikacije uporabnikov, identifikacije sej in zaključevanje poti (ang. path completion). Zaključevanje poti je hevristična metoda dodajanja manjkajočih zahtev v poti na podlagi strukture spletne strani.

Berendt in sod. [9] so se osredotočili na primerjavo med različnimi hevristikami rekonstrukcije sej. Naredili so ogrodje za primerjavo, določili mere in preizkusili model v praksi. Proces osejevanja so podobno kot Cooley [20] razdelili na dva dela: identifikacija uporabnikov in rekonstrukcija sej. V okviru identifikacije uporabnikov se uporabniku pripišejo vse njegove zahteve. V fazi rekonstrukcije sej pa se zaključeno zaporedje zahtev uporabnika razbije na več sej. Pri tem si pomagajo s hevristikami rekonstrukcije sej. Uporabili so časovno naravnano (time-oriented heuristic) in navigacijsko naravnano hevristiko (navigation-oriented heuristic) oz. njeno podvrsto z uporabo informacije o predhodno obiskani strani (ang. referrer).

Pri časovno naravnani hevristiki med dvema zahtevama v isti seji ne sme miniti preveč časa Φ. Imejmo dve zaporedni strani p in q s časoma obiska t_p in t_q ter največji časovni zamik Φ. Če je $t_q - t_p \geq \Phi$, potem se v q začne nova seja, sicer se stran q doda stari seji. Uporabili so tudi različico časovne hevristike, kjer celotna seja ne sme trajati več kot Φ časa kar pomeni, da je stran q sestavni del trenutne seje, če velja $t_q - t_0 < \Phi$. Pri hevristiki z uporabo referer informacije se q doda stari seji, dokler je referer isti ali pa podatek o referer polju manjka in velja $t_q - t_p \leq \Phi$. V nasprotnem primeru se začne v q nova seja.

Khasawneh in Chan [42] sta nadaljevala delo [9]. Namesto časovne in navigacijske hevristike sta za rekonstrukcijo sej iz zaporedja uporabnikovih obiskov strani uporabila ontologijo spletnega mesta. Zaporedje sta razdelila na podzaporedja s pomočjo znanja o spletni strani. Ontologijo spletnega mesta sta definirala kot trojico $W = (P, L, F)$, kjer P predstavlja množico vseh spletnih strani, L množico povezav med stranmi in F množico funkcij (nalog) sistema. Vsaka funkcija je sestavljena iz dveh ali več spletnih strani in predstavlja zaključeno nalogo, ki jo uporabnik izvaja. Na podlagi nalog potem avtorja določita točke preloma začetnega zaporedja strani.

Perkowitz in Etzioni [70] sta v članku z naslovom *Prilagodljive spletne strani* (ang. Adaptive Web Sites) predlagala samodejno spreminjanje in prilagajanje vsebine spletne strani uporabniku. Spremembe bi temeljile na podatkih, naučenih iz preteklega obnašanja uporabnikov. Avtorja sta predlagala algoritem PageGather, ki generira oznake tistih strani, ki se v preteklem klikotoku največkrat pojavljajo neposredno zaporedno. Na podlagi tega bi lahko ocenili zgradbo spletne strani in izvedli ustrezne prilagoditve.

3.2.3 Gručenje uporabnikov

Razvrščanje uporabnikov v gruče (ang. clusters) glede na njihove akcije na spletnem mestu je eden izmed ključnih izzivov na področju ugotavljanja obnašanja obnašanja uporabnikov (ang. web usage mining). Klikotoki različnih uporabnikov pogosto sledijo točno določenim vzorcem. Zavedanje tega, vključno s podatki o vzorcih, nam lahko pomaga pri zagotavljanju prilagojene vsebine uporabnikom. Možnosti za uporabo v poslovnem okolju so velike, še posebej na področju portalov, spletnih trgovin (ang. e-tailers) in ponudnikov prilagojenih vsebin uporabniku.

Glavne skupine procesov za napovedovanje temeljijo na odkrivanju vzorcev v podatkih o sprehajanju uporabnikov po spletnem mestu. Glavne metode za odkrivanje vzorcev so zaporedni vzorci (ang. sequential patterns), asociacijska pravila, markovski modeli in gručenje.

Lu in sod. [56] se v delu ukvarjajo z odkrivanjem vzorcev obnašanja spletnih uporabnikov. Predlagali so različico implementacije uporabnikove željene poti po spletnem mestu z imenom Significant Usage Pattern (SUP). SUP so željene poti, pridobljene iz abstraktnih gruč podatkov o klikotoku, ki imajo večje verjetnosti pojavitve od ostalih in se lahko začnejo z neko točno določeno spletno stranjo. Vsaka gručo spletnih sej so predstavili z markovskim modelom. Novost pristopa je upo-

raba gručenja nad abstraktnimi spletnimi sejami s tehniko izdelave gruč v dveh fazah. Najprej se izdela matrika podobnosti na podlagi podobnosti sekvenc abstraktnih gruč nato pa izdelajo nove gruče na podlagi matrike podrobnosti. Z izvedbo gručenja nad abstraktnimi sejami z večjo verjetnostjo dobimo skupine uporabnikov z istimi željami.

Z izzivom razvrščanja uporabnikov v skupine glede na njihove interakcije na spletnem mestu se ukvarjata tudi Banerjee in Ghosh [7]. Predlagala sta nov in učinkovit algoritem za razvrščanje spletne uporabnike v gruče na podlagi funkcije najdaljših skupnih podzaporedij (Longest Common Subsequence – LCS) njihovih klikotokov. Pri razporejanju v gruče na podlagi podobnosti LCS so vzeli v ozir tako opravljeno pot po spletnem mestu kot tudi čas zadrževanja uporabnika na posameznih spletnih straneh. Veliko število strani spletnega mesta povzroči, da je veliko število najdaljših skupnih podzaporedij majhno ali enako 0, zato so spletne strani najprej združili v skupine, imenovane koncepti, glede na vsebinsko podobnost. Poti skozi spletno mesto so nato preoblikovali glede na določene koncepte in izvedli razporejanje v gruče.

3.2.4 Uporaba markovskega modela

Dietterich [25] opisuje metode za strojno učenje na podatkih, ki so organizirani kot zaporedja elementov. Tak primer predstavlja tudi klikotok uporabnika na spletnem mestu. Predlagal je nekaj vodilnih metod za uporabo na tem področju. Med drugim se posveti tudi skritim markovskim modelom.

Cadez in sod. [12] so se osredotočili na vidike vizualizacije vzorcev prehajanja uporabnikov med spletnimi stranmi. Predstavili so nov pristop za predstavitev poti uporabnikov spletnega mesta. V pristopu so uporabnike najprej razvrstili v gruče glede na podobne poti, nato pa poti znotraj gruč ustrezno grafično prikazali obnašanje uporabnikov. Za razporejanje uporabnikov v gruče (ang. clusters) so uporabili gručenje na osnovi modela. Gručenje na osnovi modela predpostavlja, da so bili podatki generirani na osnovi modela in poskuša odkriti prvotni model iz generiranih podatkov. Za gručenje uporabnikov so uporabili mešanico različnih markovskih modelov prvega reda. Parametre markovskih modelov za vsako gručo so določili s pomočjo algoritma EM (ang. Expectation-Maximization). Za predstavitev podrobnosti postopka so razvili orodje WebCANVAS, ki omogoča predstavitev primerkov vsake gruče kot barvno ponazorjeno zaporedje kategorij spletnih strani. Preučili so tudi prilagodljivost (ang. scalability) algoritma EM za gručenje zaporedij. Pokazali so skoraj linearno prilagodljivost tako v številu zaporedij kot številu gruč.

Avtorji se osredotočajo na predstavitev in vidike gručenja, manj pa na različne probleme pri predprocesiranju podatkov v dnevnikih spletnih strežnikov. Kot osnovo so vzeli osejene podatke, kjer ima vsaka spletna stran že določeno kategorijo.

Sarukkai [82] navaja, da z rastjo spleta narašča potreba po dobrih modelih za analizo poti in predvidevanje naslednjih korakov spletnih uporabnikov. Primeren

model za to se mu zdijo markovske verige, ker jih lahko statistično ocenjujemo, so prilagodljive in razširljive. Markovske verige so torej primerno orodje za uporabo pri predvidevanju naslednje strežnikove HTTP zahteve, predvidevanje naslednje zahteve uporabnika in generiranje vodičev po spletnih straneh (zaporedij strani). Markovske verige je za zgornje probleme uporabil nad testnimi podatki. Dosegel je vzpodbudne rezultate. Nasledno zahtevo spletnega strežnika je napovedal 50% pravilno, naslednjo zahtevo uporabnika pa z več kot 60% natančnostjo. Predstavil je nov algoritem za generiranje vodičev po spletnih straneh s pomočjo markovskih verig (TUMMs). Model omogoča sprotno posodabljanje s podatki novih spletnih uporabnikov. Sarukkai predlaga, da je bi razporejanje uporabnikov v gruče izboljšalo rezultate, vendar tega ne implementira.

Članek se posveča tudi postopku predprocesiranja podatkov iz dnevnika spletnega strežnika. Navedeni so koraki od ločevanja pomembnih od nepomembnih podatkov, postopka osejevanja glede na številko IP do odstranjevanja vnosov spletnih pajkov.

Z idejo prilagodljivih spletnih strani so se ukvarjali tudi Zhu in sod. [108] z namenom pomoči porabnikom pri navigaciji po spletnem mestu. V primerjavi s [70] so naredili dve ključni izboljšavi: povezave med stranmi so vzpostavili na sosledju strani preteklih obiskov uporabnikov, zmanjšali so tudi vpliv strežnikov proxy na kakovost podatkov. Za predvidevanje naslednjih klikov uporabnika so uporabili markovski model. Markovski pristop, ki ga je uporabil Sarukkai [82] so še izboljšali. Predlagali so kombinacijo dveh pristopov izboljšave učinkovitosti markovskega modela za predvidevanje klikov. Prvi pristop je zmanjšanje velikosti matrike prehodov s pomočjo stiskanja s Spearovim algoritmon. Velikost matrike prehodov so zmanjšali z združitvijo strani s podobnimi verjetnostmi prehodov. V splošnem je tak model manj natančen, vendar tudi prostorsko manj zahteven. Za dodatno izboljšanje učinkovitosti napovedovanja so uporabili metodo najdaljše poti (ang. max. forward path) pri kateri so iz zaporedij učne množice odstranili vse zanke. S tem so zmanjšali učinek vračanj uporabnikov na prejšnje spletne strani.

Desphande in sod. [23] navajajo, da pomen predpomnjenja in napovedovanja najverjetnejših spletnih strani postaja vse večji zaradi potrebe po prilagajanju vsebine uporabniku in vplivanja na uporabnikovo izkušnjo pri brskanju. Markovski modeli in izpeljanke so se izkazali kot primerno orodje za reševanje izzivov na tem področju. Markovski modeli višjega reda nudijo višje točnosti napovedovanja, vendar so zelo zapleteni zaradi velikega števila stanj, ki povečuje prostorske zahteve in čas obdelave. Desphande in sod. so predstavili tehniko z imenom *Selective markov models* za izbor delov markovskih modelov različnih redov, tako da ima ciljni model precej manjšo prostorsko zahtevnost in ohranjeno primerljivo točnost napovedovanja. Za osnovo so vzeli shemo z naborom markovskih modelov do k-tega reda z imenom *All-K^{th} markov model*. Pri tej shemi za vsako napoved poskusimo oporabiti markovski model najvišjega reda. Če v njem ni pokritja za ta primer, vzamemo model nižjega reda itn. Model rešuje problem pokritja, ne rešuje pa problema velike prostorske kompleksnosti.

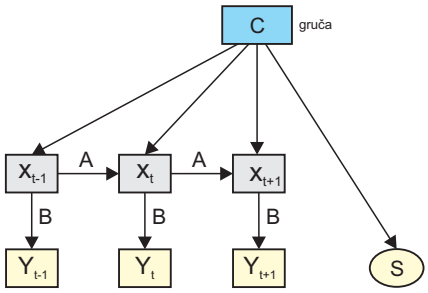
Problem prostorske kompleksnosti so rešili s pomočjo obrezovanja modela. Opisani so trije načini obrezovanja: rezanje glede na podporo, rezanje glede na zaupanje in rezanje glede na napake. Pri prvem načinu se obrežejo vsa stanja, ki nimajo dovolj velike podpore. Pri drugem načinu odstranimo stanja, kjer je verjetnost najpogostejših akcij različna od verjetnosti drugih akcij. Rezanje glede na napake poreže tista stanja v višjenivosjkih modelih, ki imajo manjšo točnost napovedovanja kot pripadajoča stanja v modelih na nižjih nivojih.

Modeliranje in napovedovanje uporabnikovih poti po spletnem mestu vključujejo kompromise med zapletenostjo modela in točnostjo napovedovanja. Pitkow in sod. [72] so se ukvarjali z izzivom, kako zmanjšati zapletenosti modela in ob tem ohraniti točnost napovedovanja. Tehnika vključuje združitev dveh metod: metode ugotavljanja pomembnejših vzorcev spletnega brskanja in identifikacije najdaljših ponavljajočih se zaporedij (ang. LRS longest repeating sequences). Metodo so primerjali z dvema različnima predstavitvama markovnskih modelov. Markovski modeli vsebujejo podatke o vseh zaporedjih v učni množici, modeli LRS pa vsebujejo samo najpogostejše uporabljena podzaporedja, kar zelo zmanjša prostorsko zahtevnost (za več kot 90% v primerjavi z markovskimi modeli višjih redov). Pokazali so, da uporaba LRS ne pomeni nujno tudi večjega zmanjšanja točnosti tako napovedovanja naslednjih uporabnikovih klikov kot tudi iskanja ujemanja za bodoče uporabnike. V primerjavi z markovskim modelom prvega reda je bil model LRS enako dober pri tretjino manjši prostorski zahtevnosti. Primerjali so tudi markovske modele višjih redov (All-Kth) z modeli LRS višjih redov in dobili zelo podobne rezultate ob znatno nižji prostorski zahtevnosti.

Z napovedovanjem naslednjih najverjetneje zahtevanih spletnih strani se ukvarjajo tudi Sen in sod. [84]. Obravnavajo modele, ki opisujejo, kako obiskovalci brskajo po spletnem mestu. Osredotočili so se na eno samo spletno mesto. Markovski model so uproabili za generiranje simuliranih dostopov spletnih obiskovalcev. Markovski model prvega reda se je izkazal za neprimernega za scenarij, kjer so generirana zaporedja klikov uporabnikov daljša od tistih iz učne množice. Bolje so se izkazali markovski modeli drugega reda, vendar predstavljajo pomembno povečanje časovne in prostorske zahtevnosti. Raziskali so tudi pristop z mešanico več markovskih modelov, s pomočjo katerih so dosegli način razvrščanja spletnih strani v gruče. Ta pristop se je izkazal za obetavnega, obenem pa je zahteval mnogo manj prostora za shranjevanje modela. Avtorji so preučili še Bayesov pristop s pomočjo Dirichletove apriorne verjetnosti povezav, ki so na voljo uporabnikom na vsakem koraku pri potovanju po spletnem mestu, vendar podrobnosti implementacije niso navedene. Navedena je tudi primerjava posameznih pristopov na primeru predpomnjenja spletnih strani (ang. web page caching).

Ypma in sod. [106] so za modeliranje klikotoka spletnih uporabnikov predlagali uporabo mešanice skritih markovskih modelov (HMM). Grafični model za posamezno gručo prikazuje slika 3.3. Avtorji so razširili delo avtorjev Cadez in sod. [12] in predstavili metodo za razporejanje spletnih uporabnikov v gruče glede na njihove vzorce obnašanja in nekatere dodatne podatke o uporabnikih. Kategorizacija se iz-

vede glede na vhodne podatke same brez potrebe po ročni kategorizaciji, ki je nerodna in okorna. Za učenje mešanice skritih markovskih modelov v gruče so uporabili algoritem EM in pokazali, da lahko za boljše razporejanje uporabnikov v gruče uporabimo dodatne statične podatke o uporabnikih. To je bil tudi glavni namen članka. Upoštevali so problem prevelike odvisnosti modela od konkretnih (ang. overfitting) in zmanjšali čas potreben za procesiranje modela. Model so preverili na umetnih in dejanskih podatkih iz dnevnikov spletnega strežnika danske spletne strani. Prikazan je primer učenja mešanice modelov HMM na umetnih podatkih in prikaz vzorcev, ki sovpadajo s pričakovanimi. Na primeru realnih podatkov so pokazali, da s pomočjo modela lahko dobimo koristne in razumljive kategorije uporabnikov.



Slika 3.3: Grafična predstavitev modela mešanice skritih markovskih modelov za gručo C. Spremenljivke Y_t in S predstavljajo opazovana stanja, kategorije X_t pa skrita stanja. S označuje dodatne statične uporabniške podatke. Skrita stanja so odvisna od gruče C, opazovana stanja in S pa ne. Matrika A skrbi za prehode med skritimi stanji, matrika B je emisijska matrika.

Levene in Loizou [49] sta predstavila teoretski model markovske verige za analiziranje vzorcev obnašanja uporabnikov na spletu, ki temelji na informaciji, pridobljeni iz tipične poti uporabnika skozi spletno mesto. Podali so teoretsko podlago za entropijo markovskega modela, ki modelira navigacijo uporabnikov po spletnem mestu. Nakazali so možne aplikacije algoritma na področju odkrivanja zakonitosti v spletnih podatkih. Entropija markovske verige spletne poti uporabnika ali skupine se lahko uporabi za izračun verjetnosti tipičnih poti uporabnikov, za aplikacijo tehnik odkrivanja zakonitosti v podatkih, lahko pa bi jo uporabili tudi v Googlovem algoritmu PageRank. Model, ki uporablja markovske verige prvega reda, so tudi razširili na uporabo markovskih verig višjih redov.

Eirinaki in sod. [31] predlagajo združitev uporabe markovskih modelov z metodami analize povezav med stranmi z namenom izboljšanja točnosti napovedovanja. Navajajo, da predstavlja uporaba izključno verjetnostnih modelov na osnovi pretekle uporabe določene pomankljivosti v povezavi z neuporabo podatkov o strukturi spletnega mesta, saj lahko pride do podcenjenosti določenih poti. Predstavili so mešan verjetnostni model, ki uporablja algoritem PageRank za določanje apriornih verjetnosti strani glede na njihov pomen v načrtu strani. Na podlagi poskusov so pokazali, da pristop nudi objektivnejše in lažje predstavljive rezultate.

Anderson in sod. [5] predstavljajo relacijski markovski model (RMM), posplošitev markovskega modela, ki predstavlja stanja sistema z relacijskimi predikati. Takšno predstavitev s pridom izkorišča za boljše učenje in sklepanje. Podana je natančna definicija modela RMM in opis načina ocenitve vrednosti prehodov med stanji. Med številnimi možnimi domenami za uporabo modela so prikazali način uporabe za primer ugotavljanja obnašanja spletnih uporabnikov. Pri modeliranju obnašanja spletnih uporabnikov z RMM so pokazali številne prednosti v primerjavi s klasičnim markovskim modelom. Primerjali so številne implementacije RMM in izbrali najboljše. Ugotovili so, da je RMM primerna alternativa klasičnemu markovskemu modelu, ker se redko obnese slabše, v določenih primerih pa se lahko odreže veliko boljše. Navajajo, da se RMM odreže dobro v primeru redke matrike prehodov stanj. V primeru dovolj velike količine podatkov pa ne nudi prednosti pred klasičnim markovskim modelom.

3.2.5 Predvidevanje naslednjih korakov uporabnika

Predvidevanje naslednjih korakov spletnega uporabnika postaja vse pomembnejše z naraščanjem števila in aktivnosti spletnih uporabnikov.

Gunduz in Ozsu [38] se naslanjata na delo [7] in predlagata nov model za modeliranje obnašanja uporabnikov spletnega mesta na podlagi drevesne predstavitve podatkov. Model upošteva tako vrstni red zahtev znotraj seje kot tudi čas zadrževanja uporabnika na posamezni strani znotraj seje. Osnovna ideja je podobna kot pri [7], pristop se razlikuje v izboljšani meri podobnosti med sejami in načinu predstavitve podatkov v gruči. Uporabniške seje razvrščata v gruče na podlagi podobnosti parov sej, vsaka gruča pa je interno predstavljena kot drevo klikotoka. Drevo klikotoka za vsako gručo predstavlja obnašanje uporabnikov od vstopa do zapustitve spletnega mesta. Vsaka nova uporabniška seja je razporejena v gručo na podlagi mere podobnosti. Predstavljen model se lahko uporablja kot del sistema za predpripravo naslednje strani kot tudi za napovedovanje naslednje strani. Model je bil preizkušen na spletnih mestih z različno strukturo strani.

Schechter in sod. [83] se ukvarjajo z izzivom učinkovite izrabe spletnega strežnika na podlagi podatkov iz dnevnika spletnega strežnika. Na podlagi podatkov v dnevniku spletnega strežnika so izdelali profile uporabniških sej. Predstavljen je postopek za učinkovito generiranje profilov uporabniških sej. Pokazali so, da je na podlagi teh profilov mogoče s presenetljivo visoko verjetnostjo dinamično generirati naslednjo stran. Ker so spletni strežniki običajno procesorsko relativno malo obremenjeni predlagajo implementacijo takšne rešitve. Če so zahteve pravilno predvidene, se zelo zmanjšajo zakasnitve pri dostavljanju vsebine uporabniku.

Di Scala in sod. [24] so zgradili bayesov hierarhični model za ovrednotenje strani spletnega mesta. Model omogoča ugotavljanje, ali ima določena spletna stran ustrezno strukturo in je ustrezno povezana z ostalimi stranmi spletnega mesta. Predpostavili so, da je možno obnašanje uporabnikov na spletnem mestu v celoti opisati z verjetnostmi prehoda med stranmi. Na klikotok so torej gledali kot

na markovsko verigo v s končnim številom stanj (ang. finite-state-space Markov chain). Za verjetnosti prehodov med stranmi so implementirali množico hierarhičnih Gaussovih apriornih porazdelitev za logit funkcijo več spremenljivk ($logit(p) = log(p) - log(1-p)$). Določili so mero atraktivnosti za stran glede na njeno vsebino, mero povezanosti strani z drugimi zanimivimi stranmi spletnega mesta in splošno mero uspešnosti strani, ki združuje obe prejšnji. Z razvitim modelom so analizirali klikotok anonimnega evropskega spletnega mesta za distribucijo programske opreme in predstavili rezultate.

Montgomery in sod. [64] so se, podobno kot nekateri drugi avtorji [72, 84], ukvarjali z izzivom ugotavljanja naslednjih uporabnikovih akcij na spletnem mestu. Razvili so model z naslovom *dynamic multinominal probit model*, ki v primerjavi z markovskimi modeli prvega reda daje občutno boljše rezultate. Predlagan statistični model omogoča ocenitev poteka preostanka uporabnikove poti, torej tudi ocenitev, ali bo uporabnik kupil izdelek in ali bo zapustil spletno mesto v roku naslednjih petih klikov. Za domeno napovedovanja avtorji običajno uporabljajo podatke na nivoju sej. Montgomery in sod. pa so se osredotočili na klikotok na nivoju strani. Avtorji so izbrali kot osnovo prehod med dvema stranema in ne prehod med dvema skupinama strani (npr. kategorijama), ki predstavljajo zaključene akcije uporabnika. Uporabili so klikotok iz odjemalcev spletnih uporabnikov, kar ni pogosta rešitev. Model so preverili na primeru pomembnejše spletne knjigarne. Pokazali so, da se po samo 6 klikih uporabnika lahko z več kot 40% natančnostjo ugotovi, ali je obiskovalec tudi kupec.

3.2.6 Vpliv prepletenih sej na analizo klikotoka

Vsa zgoraj navedena sorodna dela, ki se ukvarjajo z domeno preučevanja obnašanja spletnih uporabnikov, se zanašajo na predpostavko, da časovno zaporedje klikov uporabnika opisuje pot uporabnika po spletni strani. Implicitno se predpostavlja, da se uporabnik sprehaja po spletni strani z odprtim enim samim brskalnikovim oknom. Časovno zaporedje klikov se torej enači z zaporedjem zahtev v seji, ki smo jo rekonstruirali iz dnevnika spletnega strežnika. Na temo problema pravilne rekonstrukcije uporabniških sej na podlagi časovnega zaporedja klikov je bilo objavljenih veliko del [20, 88]. Z razmahom brskalnikov novejše generacije, kjer je podpora več zavihkom pravilo, zgornja predpostavka ne drži nujno več. Nujno je potrebno razumeti, kako uporaba več zavihkov pri brskanju po spletu vpliva na odkrivanje vzorcev obnašanja uporabnikov in še posebej na rekonstrukcijo spletne seje uporabnika. Ne glede na vrsto spletnega mesta je predpriprava podatkov za analize o klikotoku vedno potrebna, kakovost vhodnih podatkov pa bistvena za kakovost analiz.

Viermetz in sod. [97] so se osredotočili na raziskavo brskanja po spletnem mestu z odprtimi več okni brskalnika. Prepletene seje, ki pri tem nastanejo, so poimenovali paralelne seje. Predstavili so problem in na umetnem primeru prikazali netrivialnost razvrščanja klikotoka, ki je rezultat paralelnega brskanja, v seje. Glavni

prispevki dela so trije. Predstavili so splošni model za spletno brskanje, ki pokriva tudi brskanje v več oknih naenkrat. Izdelali so postopek za detekcijo sej, ki so bile generirane z odprtimi več brskalnikovimi okni naenkrat. Na podlagi konkretnega primera so zdelali oceno dejanskega deleža takšnih sej in prišli do rezultatov, da imajo pomemben vpliv na rekonstrukcijo sej.

Predstavili so formalni model LTS (ang. Label Transition System) za predstavitev prepletenih sej. Spletno stran so predstavili kot graf, kjer vozlišča predstavljajo strani, povezave pa hiperpovezave. S tem modelom je mogoče opisati tako prepletene kot neprepletene seje. Neprepleteno sejo so s pomočjo LTS definirali kot: $LB = (\Sigma, \Lambda, \mathscr{T})$, kjer Σ predstavlja vsa vozlišča grafa, vključno z začetnim in končnim stanjem, Λ vsebuje množico akcij ϕ, ki jih uporabnik lahko naredi, $\mathscr{T}$ pa množico vseh prehodov $\Sigma \times \Lambda \times \Sigma$, možnih na spletnem mestu. Prepleteno sejo, pa so definirali kot $PB = (\Sigma, \Lambda', \mathscr{T}')$, z razširjenimi množicami Λ' in $\mathscr{T}'$. Množici vseh prehodov $\mathscr{T}$ je dodan prehod v novo okno brskalnika τ. Množici vseh akcij Λ pa je dodana menjava med odprtimi okni σ brskalnika.

Pri rekonstrukciji uporabniških poti spletnega mesta so uporabili zgoraj predstavljen formalni model LTS. Vsako uporabniško sejo so v postopku rekonstrukcije predstavili kot drevo klikotoka (ang. clicktree). Tako drevo vsebuje vse možne poti od korena do listov, ki bi jih uporabnik potencialno lahko naredil. Povezave drevesa lahko zavzemajo eno izmed možnosti: prehod med dvema stanjema ϕ, odprtje novega okna z vsebino τ ali menjava med dvema odprtima oknima brskalnika σ. Na vsako pot grafa lahko gledamo kot na potencialni klikotok uporabnika. Drevo klikotoka za vsako sejo so potem porezali (ang. prune) v skladu z strukturnimi omejitvami spletnega mesta (načrt spletnega mesta). Porezano drevo predstavlja iskalni prostor za sejo, ki jo je uporabnik naredil.

Na podlagi analize dreves klikotokov za seje so določili spodnjo in zgornjo mejo števila sej, ki bi lahko bile prepletene. Pri najmanj 4% sej je izkazalo, da so gotovo prepletene.

3.3 Razpletanje prepletenih sej

Pregled dosedanjega dela na področju klikotoka kaže, da je bilo to področje v zadnjih desetih letih zelo zanimivo za raziskovalce. V veliko delih je bila v ospredju prav skrb za čimbolj verno rekonstrukcijo uporabniških sej z namenom čimvečje kakovosti podatkov. Prepletene seje lahko kvarno vplivajo na kakovost podatkov o klikotoku in predstavljajo izziv za razvoj novih postopkov rekonstrukcije uporabniških sej. Viermetz in sod. [97] so v svojem delu predstavili problem prepletenih sej. Podrobno so opisali problem in ocenili delež prepletenih sej znotraj klikotoka, niso pa podali konkretnih metod za razpletanje.

V disertaciji smo se posvetili problemu razpletanja prepletenih sej. V naslednjem razdelku bomo problem razpletanja najprej osvetlili s teoretičnega vidika.

3.4 Kombinatorični rezultati

3.4.1 Uvod

V tem razdelku se bomo posvetili kombinatoričnim rezultatom. Preden razvojem metod smo si problem razpletanja sej osvetlili s teoretične plati. Ukvarjali smo se z vidiki prepletanja in razpletanja sej ter določili formule za izračun vseh različnih možnosti. Definirali smo formulo za izračun števila vseh možnih prepletov in pokazali, kako narašča z večanjem dolžin čistih sej. Našli smo povezavo med število možnih kombinacij čistih sej v prepletu in Bellovimi števili. Preučili smo obnašanje uporabnika in določili verjetno število vsebovanih sej v prepletu. V ta namen smo določili formulo za število možnih razpletov v vnaprej znano število razpletov. S pomočjo izračunov smo pokazali na zapletenost problema razpletanja sej, ki je večji kot kaže na prvi pogled.

3.4.2 Različni prepleti sej

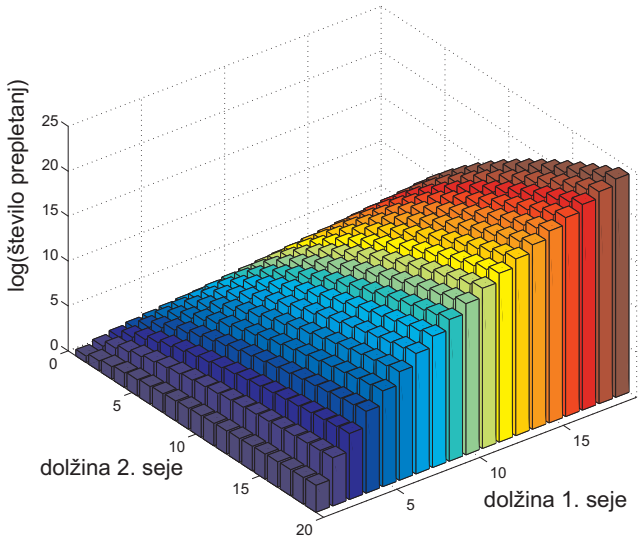
Predpostavimo, da uporabnik ob istem času vodi dve uporabniški seji. Vsaka seja je sestavljena iz zaporedja strani ustrezne dolžine. $S_1 = \{s_1, s_2, s_3\}$ Dolžina prve seje je n, dolžina druge seje pa k. $S2 = \{t_1, t_2\}$ Uporabnik lahko izvaja operacije dostopanja do strani tako v prvi kot drugi seji v različnem zaporedju. Prepletena seja, ki pri tem nastane, je dolžine $n+k$. Posamezne strani iz prve seje S_1 in druge seje S_2 se lahko nahajajo na različnih mestih v prepleteni seji. Upoštevati moramo, da je zaporedje strani prve in druge seje v prepleteni seji, enako zaporedju strani v neprepleteni seji. Pojavi se vprašanje, na koliko različnih načinov lahko uporabnik izdela prepleteno sejo, če natančno poznamo njegovo zaporedje strani v prvi in drugi seji. Koliko je možnih različnih prepletov obeh sej. Na primer, če imamo dve seji dolžine 3 in 2, lahko naredimo 10 različnih prepletov:

```
S0 S1 S2 T0 T2
S0 S1 T0 S2 T2
S0 S1 T0 T2 S2
S0 T0 S1 S2 T2
S0 T0 S1 T2 S2
S0 T0 T2 S1 S2
T0 S0 S1 S2 T2
T0 S0 S1 T2 S2
T0 S0 T2 S1 S2
T0 T2 S0 S1 S2
```

Problem. Zanima nas, koliko različnih prepletenih sej lahko kreira uporabnik iz dveh sej dolžine n_1 in n_2, če je zaporedje strani v sestavnih sejah natančno določeno.

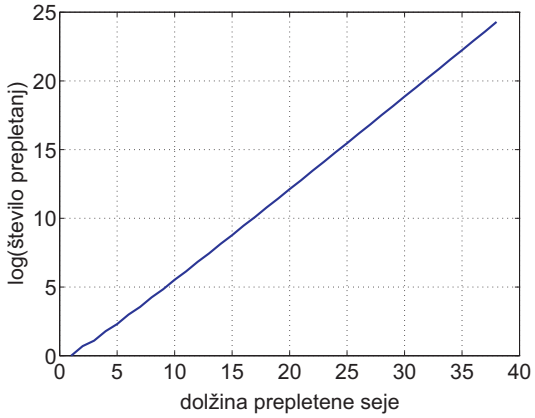
Trditev 1. Število vseh možnih prepletov lahko izračunamo s pomočjo kombinacij elementov $C = \binom{n}{k}$.

Dokaz. Število vseh možnih prepletov izračunamo s pomočjo kombinacij elementov $C = \binom{n}{k}$. Kombinacije elementov predstavljajo število načinov, kako lahko izberemo k neurejenih izidov iz n možnosti. Kombinacijo lahko definiramo tudi kot neurejeno zbirko različnih elementov določene velikosti iz dane množice [19]. Pri množici elementov M je kombinacija elemetov množice M samo podmnožica elementov iz M, kjer vrstni red ni pomemben (kot to splošno velja za množice). Dve zaporedji elementov, ki vsebujejo iste elemente v različnem vrstnem redu, se obravnava kot ista kombinacija. Element se v kombinaciji tudi ne more pojaviti dvakrat. To lastnost imenujemo tudi elementi brez ponavljanja. Elementi znotraj nepreplete-tene uporabniške seje imajo točno določen vrstni red. Imamo torej eno samo možnost izbire elementov, torej imamo množico elementov. Vrstnega reda elementov ne ločimo, kar je značilno za kombinacije elementov. Elementi neprepletene uporabniške seje torej predstavljajo množico. Število vseh možnih prepletanj dveh sej dolžine n_1 in n_2 je torej enako $C_k^n = \binom{n_1+n_2}{n_1} = \binom{n_1+n_2}{n_2}$. Razmišljamo lahko takole: iz množice elementov velikosti $n_1 + n_2$ (velikost prepletene seje) izberemo neurejeno množico elementov velikosti n_1. Število prepletov z večanjem dolžine ene ali druge seje zelo hitro narašča. Pri dolžini obeh sej 19 elementov imamo že več kot 35 milijard vseh možnih kombinacij prepletov. Na sliki 3.4 vidimo graf števila vseh možnih kombinacij prepletanj v odvisnosti od dolžine obeh sej.



Slika 3.4: Graf števila vseh možnih kombinacij prepletanj dveh sej.

Graf na sliki 3.5 prikazuje največje možno število prepletanj v odvisnosti od dolžine prepletene seje. Os X torej predstavlja vrednost $n_1 + n_2$ v razponu od 1 do 38, os Y pa vrednosti $\log \max \binom{n_1+n_2}{n_1}$.



Slika 3.5: Graf največjega števila prepletanj v odvisnosti od dolžine prepletene seje.

3.4.3 Število možnih kombinacij sej v prepletu

Ko dobimo uporabnikovo prepleteno sejo ne vemo, koliko sestavnih (čistih) sej je v njej prepleteno. V veliki večini primerov gre za preplet dveh ali treh sej. Preplet večjega števila elementarnih sej je manj verjeten zaradi narave dela uporabnika. Ponavadi uporabnik za določeno spletno mesto nima odprtih preveč oken, ker to preprosto ni praktično. Razpletom prepletenih sej z dvemi in tremu sestavnimi sejami smo se podrobneje posvetili tudi mi v tem delu, predvsem prepletenim sejam, ki vsebujejo dve seji.

Problem. Zanima nas koliko kombinacij različnih sej lahko teoretično sestavlja prepleteno sejo dolžine n oz. koliko mešanic različnih sej je lahko v zaporedju dolžine n?

Trditev 2. Prepleteno sejo lahko sestavlja B_n kombinacij različnih elementarnih sej, kjer B predstavlja Bellovo število.

Dokaz. Podseje so med sabo lahko zamešane in se prepletajo. Zaporedje elementov čistih sej je v prepleteni seji torej lahko prekinjano. Pogoj je le, da se ohranja vrstni red iz čistih sej. V množici vrstni red ni definiran, kar pomeni, da istih k različnih elementov lahko sestavlja natančno eno množico. Čiste seje, ki sestavljajo prepleteno sejo, imajo natančno določen vrstni red elementov. Iz k različnih elementov torej lahko, podobno kot pri množici, sestavimo natanko eno zaporedje. Čiste seje

so torej ekvivalentne podmnožicam pri *Bellovem številu* [80]. Bellovo število B_n predstavlja število načinov, kako lahko n elementov razporedimo v ločene neprazne podmnožice. Zanima nas samo število nepraznih sej, zato je problem popolnoma ekvivalenten Bellovemu številu.

Primer: $B_3 = 5$, ker lahko množico $\{A,B,C\}$ razdelimo v neprazne podmnožice na naslednjih 5 načinov:

$$\{A,B,C\}; \quad \{A,B\}\cup\{C\}; \quad \{A,C\}\cup\{B\}; \quad \{A\}\cup\{B,C\}; \quad \{A\}\cup\{B\}\cup\{C\}$$

Prvih nekaj Bellovih števil za n = 0...10 je: 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975 in hitro naraščajo z vsakim naslednjim številom n. Za Bellova števila velja rekurzivna formula:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k \tag{3.1}$$

Vsako Bellovo število je tudi vsota Stirlingovih števil drugega reda. $B_n = \sum_{k=0}^{n} S(n,k)$, kjer $S(n,k)$ označuje število načinov za particioniranje množice s kardinalnostjo n v natanko k nepraznih množic.

3.4.4 Število različnih razpletov v k sej

Predpostavimo, da imamo prepleteno sejo dolžine n. Prepletena seja je sestavljena iz dveh čistih sej, ki predstavljata klikotok uporabnika. V prejšnjem primeru nas je zanimalo, koliko različnih kombinacij čistih sej lahko sestavlja prepleteno sejo. Pokazali smo, da se da ta problem rešiti s pomočjo Bellovih števil. Sedaj pa nas zanima, na koliko načinov lahko prepleteno sejo razpletemo v vnaprej znano število razpletenih sej.

Problem. Na koliko načinov lahko prepleteno sejo dolžine n razporedimo v natančno k nepraznih sej.

Trditev 3. Za izračun števila razpletov seje dolžine n v natančno k nepraznih sej uporabimo *Stirlingova števila* drugega reda.

Dokaz. Stirlingova števila drugega reda določajo število particij množice velikosti n v k skupin. Ker se mora vrstni red elementov v prepletu ohraniti tudi znotraj razpletenih sej, lahko torej zgornji problem prevedemo na Stirlingova števila drugega reda. Stirlingova števila [36] tvorijo trikotno formo koeficientov, podobno kot binomski koeficienti $\binom{n}{k}$ v Pascalovem trikotniku. Obstaja več notacij za označitev Stirlingovih števil drugega reda: $S(n,k)$ [77] = $\mathscr{S}_n^{(k)}$ [2] = $\left\{ {n \atop k} \right\}$ [36]. Stirlingovo število pomeni število načinov, kako lahko množico velikosti n razporedimo v k nepraznih množic. Na primer, množico velikosti 4, lahko razporedimo v dve množici

na naslednjih 7 načinov:

$$\{1,2,3\}\cup\{4\},\{1,2,4\}\cup\{3\},\{1,3,4\}\cup\{2\},\{2,3,4\}\cup\{1\},$$
$$\{1,2\}\cup\{3,4\},\{1,3\}\cup\{2,4\},\{1,4\}\cup\{2,3\}.$$

Iz tega sledi, da $\left\{{4 \atop 2}\right\} = 7$. Notacija z uporabo zavitih oklepajev, s katerimi označujemo množice, nas pomaga spomniti na pomen oznake. Preberemo jo "n podmnožica k". Za Stirlingova števila drugega reda velja rekurenčna relacija:

$$S(n,k) = S(n-1,k-1) + k * S(n-1,k), \quad n > 0 \tag{3.2}$$

S pomočjo pravila (3.2) lahko zgeneriramo tabelo 3.2, ki prikazuje Stirlingov trikotnik za podmnožice.

Tabela 3.2: Stirlingov trikotnik za podmnožice

n	$\left\{{n \atop 0}\right\}$	$\left\{{n \atop 1}\right\}$	$\left\{{n \atop 2}\right\}$	$\left\{{n \atop 3}\right\}$	$\left\{{n \atop 4}\right\}$	$\left\{{n \atop 5}\right\}$	$\left\{{n \atop 6}\right\}$	$\left\{{n \atop 7}\right\}$	$\left\{{n \atop 8}\right\}$	$\left\{{n \atop 9}\right\}$
0	1									
1	0	1								
2	0	1	1							
3	0	1	3	1						
4	0	1	7	6	1					
5	0	1	15	25	10	1				
6	0	1	31	90	65	15	1			
7	0	1	63	301	350	140	21	1		
8	0	1	127	966	1701	1050	226	28	1	
9	0	1	255	3025	7770	6951	2646	462	36	1

Poseben primer dobimo, če je $n > 0$ in $k = 2$ [36]. V tem primeru elemente množice razdelimo na dva neprazna dela. Eden izmed teh dveh delov vsebuje zadnji element in neko podmnožico prvih $n-1$ elementov. Obstaja 2^{n-1} načinov izbire zadnje množice, ker je vsak izmed prvih $n-1$ elemenetov bodisi v tej množici bodisi ga ni. Ne smemo pa vanjo postaviti vseh elementov, ker želimo končati z dvema nepraznima deloma. Zato odštejemo 1 in dobimo: $\left\{{n \atop 2}\right\} = 2^{n-1} - 1$, kjer je $n > 0$. S pomočjo zgornjega razmisleka lahko pridemo do zgoraj navedene rekurenčne relacije. Stirlingova števila drugega reda lahko izračunamo tudi s pomočjo različnih številskih vrst [79]. Npr.:

$$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^n \tag{3.3}$$

3.5 Možni pristopi k reševanju problema razpletanja

Dinamično programiranje je pogosto uporabljena metoda za dekodiranje verjetnostnih modelov z oblikovanimi izhodi, kot so HMM, pogojna naključna polja (ang.

Conditional Random Fields – CRF) in stohastične kontekstno neodvisne gramatike (ang. Stohastic Context Free Gramatics – SCFG). Dinamično programiranje sicer ponuja algoritem za dekodiranje teh modelov v polinomskem času, vendar je to lahko še vedno prepočasno [15]. Iskanje optimalne razčlembe z uporabo SCFG zahteva $O(n)$ časa, kjer je n število žetonov na vhodu. Uporaba SCFG s časovno zahtevnostjo $O(n)$ nad zelo obsežnimi vhodnimi podatki je lahko zelo draga [98]. Tudi pri enostavnih modelih, kot je HMM, ki zahtevajo $O(n)$ časa za dekodiranje, je uporaba dinamičnega programiranja zaradi skritih konstant pri velikem številu stanj vprašljiva. Razlog tiči v dejstvu, da časovna zahtevnost uporabe HMM raste s kvadratom števila stanj.

Zaradi tega je bilo predlaganih veliko alternativ dinamičnemu programiranju, kot je snopovno iskanje (ang. Beam search), dekodiranje po sistemu najprej najboljši (ang. best-first decoding) [15] in A* [46]. Snopovno iskanje in dekodiranje po sistemu najprej najboljši ne zagotavljata *optimalne rešitve*, medtem ko A* zagotavlja optimalno rešitev, če najdemo popolno podoceno (ang. admissible underestimate) [27].

Algoritem A* uporablja za vrednost usmerjevalne funkcije seštevek cen trenutne rešitve in optimistično podoceno do končne rešitve. Sheme za prioritetno iskanje morajo v vsakem koraku izvesti dodano delo v primerjavi z dinamičnim programiranjem. Zato mora usmerjevalna funkcija porezati pomemben del preiskovalnega prostora, da je učinkovita. Za učinkovito uporabo A* (t.j. hitrejši kot Viterbi), moramo torej znati hitro izračunati oceno. Ena izmed največjih ovir pri uporabi algoritma A* je tudi ta, da ne znamo vedno generirati popolne podocene in ga zato ne moremo vedno uporabiti [27].

V razdelku 3 je bila predstavljena zahtevnost problema razpletanja. Razpletanje še otežuje dejstvo, da imajo spletna mesta izredno veliko število spletnih strani. Z združevanjem strani v gruče se izgubi nekaj informacije in posledično zmanjša točnost razpletanja. Tudi če izvedemo združevanje strani v gruče, še vedno lahko dobimo število stanj reda 1000. Veliko število stanj onemogoča učinkovito uporabo HMM in Viterbijevega algoritma. Za algoritem A* smo uspeli najti hevristično funkcijo, ki je popolna, in s tem odprli pot za njegovo uporabo.

POGLAVJE 4

Metode

4.1 Uvod

Pri izdelavi postopkov za razpletanje sej smo uporabili markovske modele in metode reševanja problemov s pomočjo preiskovanja prostora stanj.

Vsak spletni uporabnik pusti sled o obiskanih straneh v dnevniku spletnega strežnika. Analizo podatkov o klikotoku zaradi velike količine podatkov težko izvajamo 'ročno'. V praksi so se izkazali kot uspešni na področju modeliranja in analize klikotoka markovski modeli [76]. Z markovskimi modeli lahko modeliramo obnašanje uporabnikov pri sprehajanju po spletnem mestu s pomočjo verjetnosti prehodov med posameznimi stranmi, ki jih izračunamo na podlagi zabeleženih preteklih uporabniških zahtev v dnevniku spletnega strežnika [31].

Pri postopku razpletanja seje naletimo na problem različnih možnih razpletov prepletene seje. Alternative je potrebno pregledati in izluščiti najbolj verjetno možnost. Preiskovanje prostora stanj nudi učinkovite algoritme za iskovanje alternativ, zato smo problem razpletanja prevedli na preiskovanje prostora stanj. V nadaljevanju poglavja bomo podrobneje predstavili obe področji.

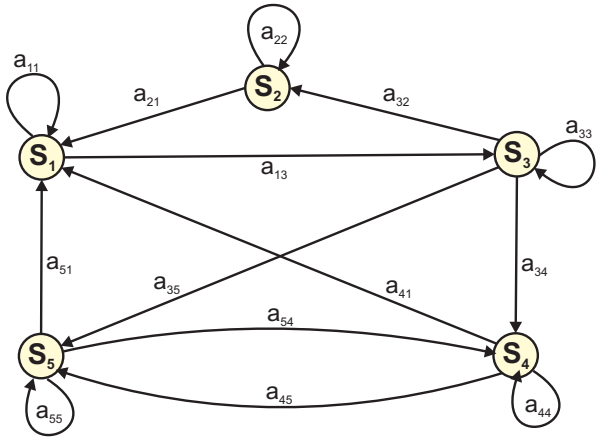
4.2 Markovski model

4.2.1 Markovski model prvega reda

Pri reševanju problemov, ki so neločljivo povezani s časovnostjo – to pomeni, da so sestavljeni iz procesa, ki se razvija v času – imamo lahko stanja v času t, na katera vpliva neposredno stanje v času $t-1$. Markovski modeli (ang. Markov Model) in skriti markovski modeli (ang. Hidden Markov Model – HMM) se veliko

uporabljajo pri reševanju takšnih problemov, na primer pri prepoznavanju govora, prepoznavanju premikanja, napovedovanju naslednjih dogodkov, ipd.

Vzemimo sistem, ki ga lahko opišemo tako, da je v vsakem trenutku v enem izmed med seboj različnih N stanj $S_1, S_2, \ldots, S_N$. Tak sistem za $N = 5$ (zaradi enostavnosti) prikazuje slika 4.1. V enakomerno razmaknjenih diskretnih časov-



Slika 4.1: Markovska veriga s 5 stanji (označenimi od S_1 do S_5) z označenimi prehodi med stanji.

nih trenutkih je sistem podvržen spremembam stanja glede na množico verjetnosti prehodov, povezanih s tem stanjem. Sistem se lahko seveda 'spremeni' (premakne) tudi v isto stanje. Časovne trenutke, povezane s spremembami stanja, označimo kot $t = 1, 2, \ldots,$. Dejansko stanje v času t pa označimo kot q_t. Popoln verjetnostni opis zgornjega sistema bi v splošnem zahteval opredelitev trenutnega stanja (v času t) kakor tudi vseh predhodnih stanj. Za poseben primer diskretne markovske verige prvega reda se verjetnostni opis skrči samo na trenutno in predhodno stanje. To zapišemo kot:

$$P(q_t = S_j \,|\, q_{t-1} = S_i, q_{t-2} = S_k, q_{t-3} = S_l, \ldots) \;=\; P(q_t = S_j \,|\, q_{t-1} = S_i) \qquad (4.1)$$

Zanimali nas bodo predvsem takšni procesi, kjer je desna stran stran enačbe (4.1) neodvisna od časa. Na ta način lahko definiramo množico verjetnosti prehodov (ang. transition probabilities) a_{ij} oblike:

$$a_{ij} = P(q_t = S_j \,|\, q_{t-1} = S_i), \;\; 1 \leq i, j \leq N \qquad (4.2)$$

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{bmatrix} \qquad (4.3)$$

kjer za koeficiente verjetnosti prehoda veljajo naslednje lastnosti:

$$a_{ij} \geq 0 \tag{4.4}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{4.5}$$

Zadnji parameter markovske verige, ki ga moramo definirati, je začetno stanje verige. Začetno stanje je lahko neko vnaprej točno določeno fiksno stanje. Lahko pa je določeno z verjetnostno porazdelitvijo nad množico stanj v obliki verjetnostnega vektorja π,

$$\pi = (\pi_1, \pi_2, \pi_3, \ldots, \pi_n) \tag{4.6}$$

kjer π_i označuje verjetnost začetnega stanja S_i. Povedano drugače:

$$\pi_i = P(q_1 = S_i), \;\; 1 \leq i \leq N \tag{4.7}$$

Zgornji stohastični proces bi lahko imenovali tudi opazovani (ang. observable) markovski model, ker je izhod procesa v vsakem trenutku kar množica množica stanj, kjer vsako stanje označuje fizičen (opazovan) dogodek. Za primer vzemimo enostaven markovski model vremena s tremi stanji. Predpostavimo, da vsak dan opoldne pogledamo, kakšno je vreme. Vreme lahko pripada enemu izmed naslednjih treh stanj:

- stanje 1: dež ali sneženje,
- stanje 2: oblačno,
- stanje 3: sončno.

Domnevamo, da vreme v dnevu t lahko opišemo z enim od zgornjih treh stanj in je matrika verjetnosti prehodov A enaka

$$A = \{a_{ij}\} = \begin{bmatrix} 0,4 & 0,3 & 0,3 \\ 0,2 & 0,6 & 0,2 \\ 0,1 & 0,1 & 0,8 \end{bmatrix} \tag{4.8}$$

Če predpostavimo, da je vreme na prvi dan (t = 1) sončno (stanje 3), se lahko vprašamo naslednje vprašanje: kakšna je verjetnost (glede na model), da bo vreme v naslednjih sedmih dneh 'sonce-sonce-dež-dež-sonce-oblačno-sonce'? Povedano bolj formalno, definiramo opazovano zaporedje O kot $O = \{S_3, S_3, S_3, S_1, S_1, S_3, S_2, S_3\}$ glede na čas $t = 1, 2, \ldots, 8$ in želimo določiti verjetnost opazovanega zaporedja O pri podanem modelu. Verjetnost lahko izrazimo kot

$$\begin{aligned} P(O\,|\,\text{Model}) &= P(S_3, S_3, S_3, S_1, S_1, S_3, S_2, S_3 \mid \text{Model}) \\ &= P(S_3) \cdot P(S_3|S_3) \cdot P(S_3|S_3) \cdot P(S_1|S_3) \\ &\quad \cdot P(S_1|S_1) \cdot P(S_3|S_1) \cdot P(S_2|S_3) \cdot P(S_3|S_2) \\ &= \pi_3 \cdot a_{33} \cdot a_{33} \cdot a_{31} \cdot a_{11} \cdot a_{13} \cdot a_{32} \cdot a_{23} \\ &= 1 \cdot 0,8 \cdot 0,8 \cdot 0,1 \cdot 0,4 \cdot 0,3 \cdot 0,1 \cdot 0,2 \\ &= 1,536 \times 10^{-4} \end{aligned}$$

V splošnem torej lahko za zaporedje stanj $(q_1, q_2, \ldots q_k)$ izračunamo verjetnost zaporedja z množenjem verjetnosti začetnega stanja $P(q_1)$ z verjetnostmi prehodov v naslednja stanja:

$$P(zaporedje) = P(q_1) \cdot \prod_{i=2}^{k} P(q_i|q_{i-1}) \tag{4.9}$$

Z uporabo danega modela lahko odgovorimo še na eno zanimivo vprašanje. Recimo, da je model v nekem določenem stanju, kakšna je verjetnost, da ostane v tem stanju natanko d dni? Verjetnost lahko ocenimo kot verjetnost opazovanega zaporedja $O = \{S_i, S_i, S_i, \ldots S_i, S_j \neq S_i\}$ pri modelu, ki je:

$$P(O\,|\,\text{Model}, q_1 = S_i) = (a_{ii})^{d-1}(1 - a_{ii}) = p_i(d) \tag{4.10}$$

Količina $p_i(d)$ se imenuje funkcija (diskretne) verjetnostne gostote trajanja d v stanju i. Ta eksponentna gostota trajanja je lastnost trajanja v stanju v markovski verigi. Glede na $p_i(d)$ lahko brez oklevanja izračunamo pričakovano število opazovanih stanj, če je to tudi začetno stanje, kot:

$$\overline{d}_i = \sum_{d=1}^{\infty} d \cdot p_i(d) \tag{4.11}$$

$$= \sum_{d=1}^{\infty} d \cdot (a_{ii})^{d-1}(1 - a_{ii}) = \frac{1}{1 - a_{ii}} \tag{4.12}$$

Iz tega sledi, da je pričakovano število zaporednih dni sončnega vremena, glede na model, enako $1/(0,2) = 5$; za oblačne dni je številka $2,5$, za dež pa $1,67$ [76].

Matriko prehajanja stanj A lahko določimo na veliko načinov. Vrednosti prehodov npr. lahko določimo s pomočjo znanja o določenem poslovnem problemu. Najbolj pogost pristop pa je izračun vrednosti matrike s pomočjo *učne množice* zaporedij, ko ocenimo vsak vnos a_{ij} glede na pogostost dogodka, ko akcija j sledi stanju s_i.

4.2.2 Markovski modeli višjih redov

Stohastični proces, kjer je verjetnostna porazdelitev naslednjega stanja odvisna samo od trenutnega stanja, se imenuje markovski model (MM) *prvega reda*. Če je naslednje stanje odvisno od trenutnega in predhodnega stanja, dobimo malce bolj zapleten markovski model *drugega reda*. Njegova stanja se nanašajo na vse možne pare akcij, ki jih je možno izvesti v zaporedju. Ta postopek lahko razširimo na markovske modele poljubnega (k-tega) reda. Pri markovskem modelu k-tega reda določimo naslednje stanje na osnovi k predhodnih stanj, kar ima za posledico matriko prehodov stanj A, ki vsebuje vsa možna zaporedja dolžine k.

Z markovskimi modeli višjih redov hočemo doseči višjo natančnost določanja naslednjega stanja. Markovnski modeli prvega reda v določenih primerih ne zadostujejo več, ker ne gledajo dovolj daleč v preteklost, da bi lahko razlikovali med

različnimi načini obnašanja problema, ki ga modelirajo. Na številne težave pa naletimo tudi pri markovskih modelih višjih redov. Med njimi so: (i) velika prostorska zahtevnost, (ii) majhno pokritje in (iii) včasih celo slabša natančnost napovedovanja kot pri markovskem modelu prvega reda [23].

Število stanj v markovskih modelih višjega reda narašča eksponentno s povečevanjem reda modela. Razlog za to je dejstvo, da so stanja modelov višjih redov enostavno različne kombinacije opazovanih akcij v množici. Dramatično povečanje števila stanj lahko pomembno vpliva na možnost apliciranja takega modela v praksi bodisi zaradi omejitev pri pomnilniku bodisi zaradi premajhne hitrosti napovedovanja pri zahtevanih realno-časovnih aplikacijah. Zaradi velikega števila stanj imamo lahko v testni množici primere, ki nimajo pripadajočih stanj v markovskem modelu višjega reda. To pa pomeni manjšo pokritost. Ker je matrika prehodov A veliko večja in raste eksponentno z redom markovskega modela, bi se moralo na tak način povečevati tudi množica učnih podatkov za markovski model, kar pa ni vedno možno doseči. V primeru premajhne pokritosti, markovski modeli višjega reda lahko slabše napovedujejo stanja.

Primer

Primer markovskega modela višjega reda predstavimo na primeru napovedovanja naslednje spletne strani. Vhod za markovski model predstavljajo spletne seje, kjer je vsaka seja predstavljena iz obiskov spletnega mesta. V tem primeru *akcije* markovskega modela predstavljajo različne strani na spletnem mestu, *stanja* pa ustrezajo zaporednim dostopom strani dolžine k, ki smo jih opazovali v različnih sejah. V primeru markovskega modela prvega reda so stanja samostojne strani, v primeru modela drugega reda pa stanja ustrezajo vsem parom zaporednih strani, itn.

Za učenje testnega markovskega modela uporabimo učna zaporedja na sliki 4.2. Če vzamemo kot primer markovski model prvega reda in spletno sejo $SS_2 = (\{P_3;P_5;P_2;P_1;P_4\})$, vsako stanje predstavlja eno stran. Prva stran P_3 pripada stanju s_3. Ker stran P_5 sledi stanju s_3, se bo v matriki prehoda element a_{35} povečal za 1. Podobno bo naslednje stanje s_5 in vnos t_{52} bo ustrezno ažuriran. V primeru markovskega modela višjega reda je stanje sestavljeno iz več kot ene akcije. Za model drugega reda za sejo SS_2 je prvo stanje sestavljeno iz strani $\{P_3,P_5\}$ in ker stran P_2 sledi stanju $\{P_3,P_5\}$, bo v matriki prehodov stanj ažurirano mesto v tabeli glede na stanje $\{P_3,P_5\}$ in stran P_2. Matrika prehodov za markovski model drugega reda testnega primera ima veliko več stanj, kot je prikazano na sliki 4.2. Prikazana so samo tista stanja, ki so udeležena pri učenju matrike.

4.3 Preiskovanje prostora stanj

Na področju umetne inteligence se problem pogosto modelira kot *prostor stanj*, to je množica stanj, v katerih se problem lahko nahaja in prehodov med njimi. Prostor stanj je graf, katerega vozlišča ustrezajo problemskim situacijam, danemu problemu

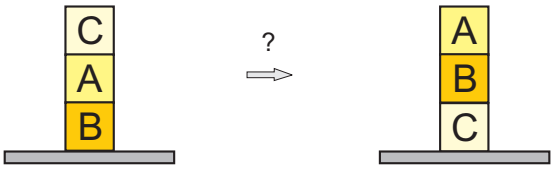
Spletne seje
$SS_1 = \{P_3, P_2, P_1\}$
$SS_2 = \{P_3, P_5, P_2, P_1, P_4\}$
$SS_3 = \{P_4, P_5, P_2, P_1, P_5, P_4\}$
$SS_4 = \{P_3, P_4, P_5, P_2, P_1\}$
$SS_5 = \{P_1, P_4, P_2, P_5, P_4\}$

1.reda	P_1	P_2	P_3	P_4	P_5
$s_1 = \{P_1\}$	0	0	0	2	1
$s_2 = \{P_2\}$	4	0	0	0	1
$s_3 = \{P_3\}$	0	1	0	1	1
$s_4 = \{P_4\}$	0	1	0	0	2
$s_5 = \{P_5\}$	0	3	0	2	0

2.reda	P_1	P_2	P_3	P_4	P_5	**2.reda**	P_1	P_2	P_3	P_4	P_5
$\{P_1, P_4\}$	0	1	0	0	0	$\{P_3, P_5\}$	0	1	0	0	0
$\{P_1, P_5\}$	0	0	0	1	0	$\{P_4, P_2\}$	0	0	0	0	1
$\{P_2, P_1\}$	0	0	0	1	1	$\{P_4, P_5\}$	0	2	0	0	0
$\{P_2, P_5\}$	0	0	0	1	0	$\{P_5, P_2\}$	3	0	0	0	0
$\{P_3, P_2\}$	1	0	0	0	0	$\{P_3, P_4\}$	0	0	0	0	1

Slika 4.2: Matriki prehajanja stanj A za vzorčne spletne seje markovskega modela 1. in 2. reda. Povzeto po [23].

pa pri tem ustreza iskanje poti v tem grafu. Reševanje problemov zahteva preiskovanje grafa, pri tem pa naletimo na problem alternativ. Dve osnovni strategiji za preiskovanje alternativ sta iskanje v globino in iskanje v širino. V tem delu se bomo osredotočili na iskanje v širino. Preiskovanje prostora stanj predstavimo na primeru problema preurejanja sklada kock. Imamo sklad iz kock na levi strani slike 4.3. Naloga je poiskati načrt za preureditev sklada kock, kot je prikazan na desni. Premeščati smemo samo po eno kocko naenkrat. Kocko smemo prijeti samo, če na njej ni ničesar. Na mizi imamo lahko naenkrat največ tri sklade. Za izdelavo načrta moramo najti zaporedje premikov, na podlagi katerega dosežemo zahtevano preureditev. Nalogo si lahko predstavljamo kot problem preiskovanja vseh možnih



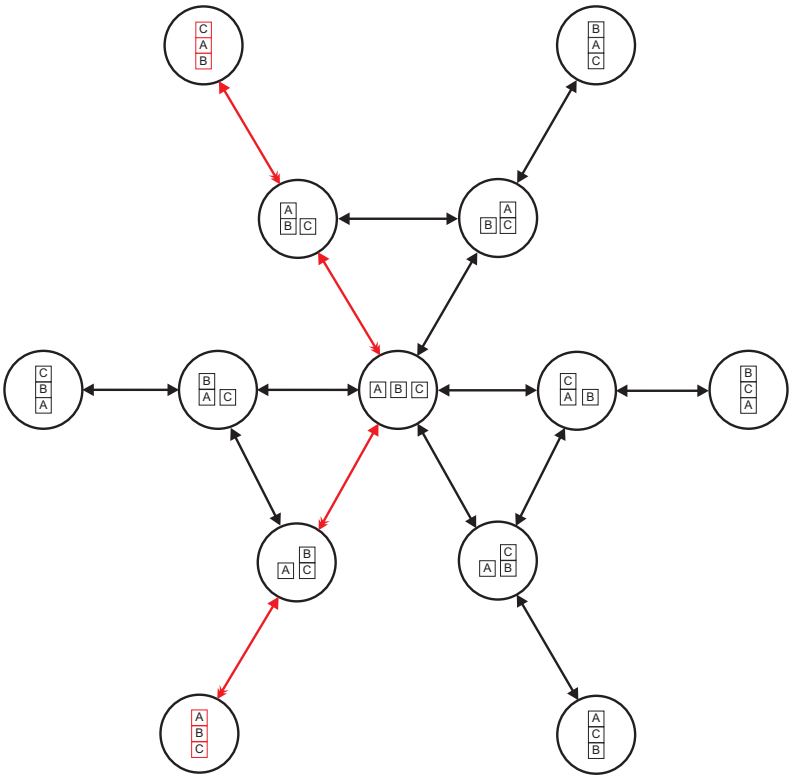
Slika 4.3: Problem preurejanja kock. Povzeto po [10].

alternativ. V začetnem položaju imamo na voljo samo eno možnost: postavi kocko C na mizo. Potem, ko je kocka C na mizi, imamo tri možnosti: (i) postavi kocko A na mizo, (ii) postavi kocko A na C (iii) postavi kocko C na kocko A. Poleg tega lahko prestavimo kocko C iz mize na mizo, kar pa ne spremeni položaja in zato nima smisla. Na tem primeru lahko vidimo, da imamo pri takih problemih opraviti z dvema konceptoma:

1. problemske situacije,

2. legalnih potez ali akcij, ki preoblikujejo problemske situacije v nove problemske situacije.

Problemske situacije in možne poteze tvorijo usmerjen graf, ki ga imenujemo *prostor stanj*. Prostor stanj za primer premikanja kock je prikazan na sliki 4.4. Vozlišča grafa ustrezajo problemskim situacijam oziroma *stanjem*, povezave pa legalnim prehodom med stanji. Najti rešitev pomeni poiskati pot med dano začetno situacijo (začetnim vozliščem) in neko določeno končno situacijo, ki ji pravimo tudi *ciljno vozlišče*. Prostor stanj za dani problem podaja pravila igre. Konkreten problem je



Slika 4.4: Ponazoritev prostora stanj problema premeščanja kock. Označena pot predstavlja rešitev problema na sliki 4.3.

torej dan:

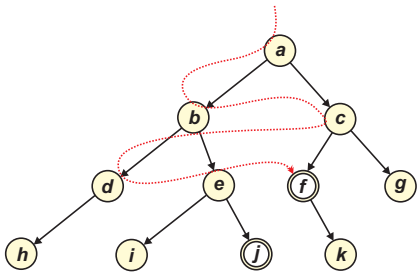
- s prostorom stanj,
- z začetnim vozliščem in
- s ciljnim pogojem (pogojem, ki ga moramo doseči). *Ciljna vozlišča* so tista vozlišča, ki zadoščajo temu pogoju.

Akcijam oziroma dovoljenim potezam lahko pripišemo cene. Na primer pri nalogi premeščanja kock lahko s cenami premikov povemo, da je nekatere kocke težje

premikati kot druge. Kadar so potezam pripisane cene, nas zanimajo najcenejše rešitve. Cena rešitve je vsota cen povezav vzdolž rešitvene poti. Tudi če cene povezav niso predpisane, imamo lahko opravka z optimizacijskim problemom. V tem primeru lahko privzamemo, da so vse cene enake 1 in je potem najcejenša pot tudi najkrajša pot.

4.3.1 Iskanje v širino

Strategija iskanja rešitve v širino (ang. breadth-first search) v nasprotju z iskanjem v globino obiskuje najprej tista vozlišča, ki so najbližja začetnemu. Na ta način se proces preiskovanja razvija bolj v širino kot v globino. Postopek iskanja v širino prikazuje slika 4.5. Iskanja v širino ne moremo implementirati tako tako preprosto kot iskanje v globino. Razlog za to je dejstvo, da moramo v vsakem trenutku voditi celo *množico* vozlišč-kandidatov in ne samo enega kot pri iskanju v globino. V to to množico spada celoten celoten spodnji rob rastočega drevesa iskanja. Toda niti ta množica vozlišč ni dovolj, če želimo iz iskanja izluščiti tudi rešitveno pot. Zato moramo namesto množice kandidatov-vozlišč voditi množico kandidatov-*poti*. Prostor stanj z danim seznamom poti-kandidatov preiščemo z iskanjem v širino



Slika 4.5: Enostaven prostor stanj: *a* je začetno vozlišče, *f* in *j* sta ciljni vozlišči. Po iskalni strategiji v širino obiščemo vozlišča v vrstnem redu: *a*, *b*, *c*, *d*, *e*, *f*. Krajšo rešitveno pot **[a,c,f]** najdemo prej kot daljšo **[a,b,e,j]**.

takole:

- če je *glava prve poti* ciljno vozlišče, potem je ta pot rešitev problema, sicer
- odstranimo prvo pot iz množice kandidatnih poti in generiramo množico vseh možnih eno-koračnih podaljšanj odstranjene prve poti. Generirano množico podaljšanj dodamo na konec seznama kandidatov in izvedemo algoritem iskanja v širino na ažurirani množici kandidatnih poti.

Programi za iskanje v širino dajejo rešitvene poti eno za drugo, pri čemer se najprej pojavijo krajše rešitve, potem pa daljše. To je pomembno, kadar nas zanimajo rešitve, optimalne glede na dolžino. Iskanje v širino vedno da najkrajšo rešitev. Če kot optimizacijsko merilo privzamemo minimalno ceno poti in ne njene dolžine,

potem iskanje v širino ne zadošča več. V tem primeru moramo uporabiti *usmerjeno iskanje*, ki si prizadeva minimizirati ceno poti.

Tipičen problem pri preiskovanju grafov je *kombinatorična zahtevnost*. Pri netrivialnih problemskih domenah je število alternativ tako veliko, da postane zahtevnost kritična. Enostavno lahko vidimo, zakaj se to zgodi. Predpostavimo, da prostor stanj predstavlja drevo, kjer ima vsako vozlišče, razen listov natanko b naslednikov. Predpostavimo, da je najkrajša pot rešitve enaka d in na nivoju d drevo nima listov. Število alternativnih poti dolžine d od začetnega vozlišča je torej b^d. Algoritem iskanja v širino bo raziskal število poti reda b^d v času $O(b^d)$. Število poti-kandidatov raste zelo hitro z njihovo dolžino, kar vodi v tako imenovano *kombinatorično eksplozijo*.

Časovna zahtevnost algoritma iskanja v širino je reda $O(b^d)$. Iskanje v širino hrani vse poti-kandidate v pomnilniku, torej je tudi prostorska zahtevnost reda $O(b^d)$.

4.3.2 Hevristično iskanje po načelu najprej najboljši

Preiskovanje grafov pri reševanju problemov tipično vodi do problema kombinatorične eksplozije zaradi hitrega naraščanja alternativ. Zato si pri rešitvi tega problema pomagamo s hevurističnim iskanjem, katerega namen je čimbolj učinkovito najti končno rešitev. Eden izmed načinov uporabe hevristične informacije o problemu je izračun številske *hevristične ocene* vozlišč v prostoru stanj. Takšna ocena vozlišča označuje, kako obetavno je vozlišče glede na doseganje ciljnega vozlišča. Glavna ideja je nadaljevanje iskanja vedno v smeri najbolj obetavnega vozlišča v množici kandidatov.

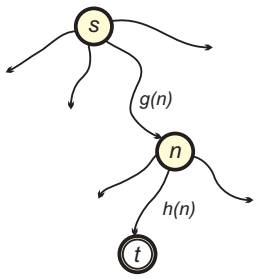
Iskanje po načelu najprej najboljši

Program za usmerjeno iskanje (ang. best-first search) lahko izpeljemo iz programa za iskanje v širino (ang. breadth-first search). Usmerjeno iskanje prav tako začnemo v prvem vozlišču in vzdržujemo množico poti-kandidatov. Iskanje v širino vedno nadaljuje iskanje z najkrajšo potjo v tej množici (to je najplitvejše vozlišče v iskalnem drevesu). Usmerjeno iskanje postopek izpopolni z uporabo hevristične ocene, ki se izračuna za vsakega kandidata v množici. Iskanje se nadaljuje v smeri kandidata, ki ima najboljšo oceno.

Predpostavimo, da je za vsako povezavo v prostoru stanj določena *cena* $c(n,n')$, ki določa ceno prehoda med vozliščima n in n'.

Hevristično cenilko (ang. heuristic estimator) bo predstavljala funkcija f, tako da za vsako vozlišče n v prostoru stanj $f(n)$ ocenjuje *'težavnost' n*. Najbolj obetavno kandidatno vozlišče je tisto, ki minimizira vrednost funkcije f. Uporabili bomo posebno sestavljeno funkcijo f, na kateri temelji dobro znani algoritem A* (A-zvezdica algoritem). $f(n)$ bo sestavljena tako, da bo ocenjevala ceno najboljše poti od začetnega vozlišča s do ciljnega vozlišča pod pogojem, da ta pot vodi skozi n. Predpostavimo, da taka pot obstaja in da je ciljno vozlišče, ki minimizira njeno

pot vozlišče t. Potem lahko cenilko $f(n)$ zgradimo kot vsoto dveh členov, kot vidimo na sliki 4.6:



Slika 4.6: Zgradba hevristične ocene $f(n)$ cene najcenejše poti od s do t preko n: $f(n) = g(n) + h(n)$. Povzeto po [10].

$$f(n) = g(n) + h(n)$$

$g(n)$ je ocena cene najboljše poti od začetnega vozlišča s do vozlišča n, $h(n)$ je ocena cene optimalne poti od vozlišča n do ciljnega vozlišča t.

Ko pri procesu preiskovanja naletimo na vozlišče n, imamo naslednji stanje: pot od s do n je že znana in ceno te poti lahko izračunamo kot vsoto povezav na tej poti. Ta pot ni nujno najboljša pot od s do n (lahko obstaja boljša pot od s do n, ki pa je proces iskanja še ni odkril), toda njena cena lahko služi kot ocena $g(n)$ minimalne cene od s do n. Drugi člen $h(n)$ je bolj težaven, ker 'sveta' med n in t do te točke še nismo raziskali. Zato je $h(n)$ pravo hevristično ugibanje, ki temelji na splošnem znanju, ki ga ima algoritem o konkretni problemski domeni. Ker je funkcija h odvisna od problema, ni splošnega načina za njeno izdelavo.

Delovanje usmerjenega iskanja si lahko predstavljamo takole. Proces iskanja je sestavljen iz več podprocesov, ki med seboj tekmujejo in vsak od njih raziskuje svojo možnost v prostoru stanj; to pomeni, raziskuje svoje lastno poddrevo. Poddrevesa imajo svoja poddrevesa: s temi se ukvarjajo podprocesi podprocesov, itn. Med tekmujočimi procesi je v vsakem trenutku aktiven le eden: tisti, ki trenutno raziskuje trenutno najbolj obetavno poddrevo; to je možnost z najnižno vrednostjo ocene f. Ostali procesi morajo počakati do takrat, dokler se ocene f ne spremenijo tako, da postane kakšna druga možnost bolj obetavna. Ta mehanizem proženja si lahko predstavljamo tudi takole: proces, ki obravnava trenutno najbolj obetavno možnost, ima na voljo določeno kvoto in ostane aktiven, dokler ta kvota ni izčrpana. Med aktivnostjo proces podaljšuje svoje poddrevo. Če pri tem naleti na ciljno vozlišče, sporoči, da je našel rešitev. Kvota je odvisna od hevristične ocene h druge najbolj obetavne možnosti.

Program za usmerjeno iskanje, ki deluje po teh principih, je inačica hevrističnega algoritma, ki je v literaturi znan pod imenom A*. Algoritem A* je eden osnovnih algoritmov umetne inteligence. Pomemben rezultat iz matematične analize tega algoritma je:

4.3.3 Popolnost

Preiskovalni algoritem je *popoln* (ang. admissible), če vedno najde optimalno rešitev; to je najcenejšo pot ob pogoju, da taka pot sploh obstaja. Program lahko poišče s samodejnim vračanjem vse rešitve. Vendar imamo tak program za popoln, če je *prva* rešitev, ki jo najde, optimalna. Naj za vsako vozlišče n v prostoru stanj, $h^*(n)$ označuje ceno optimalne poti od n do najbljižjega ciljnega vozlišča. Izrek o popolnosti algoritma A* pravi: algoritem A*, ki uporablja hevristično funkcijo h, tako da, za vsako vozlišče n v prostoru stanj velja

$$h(n) \leq h^*(n)$$

je popoln. Popolne hevristike so po naravi optimistične, saj mislijo, da je cena rešitve problema manjša, kot je v resnici.

Ta rezultat ima velik praktičen pomen. Tudi če ne vemo natančne vrednosti h^* je dovolj, da najdemo spodnjo mejo h^* in jo uporabimo kot h v A*. To je dovolj velika garancija, da bo A* našel optimalno rešitev. Obstaja trivialna spodnja meja, in sicer:

$$h(n) = n; \text{ za vse } n \text{ v prostoru stanj}$$

Ta funkcija dejansko že tudi zagotavlja popolnost, žal pa ta funkcija pri $h = 0$ nima nobene hevristične moči in ne zagotavlja usmerjanja iskanja. A^*, ki uporablja $h = 0$, deluje podobno kot iskanje v širino. V primeru, da so vse cene povezav v prostoru stanj med (n,n') enake $c(n,n') = 1$, potem je A^* s $h = 0$ ekvivalenten iskanju v širino. Če h nima hevristične moči, se kombinatorična zahtevnost spet poveča. Zato je najboljša takšna funkcija h, ki je spodnja meja funkcije h^* (da zagotovi popolnost) in je tudi čim bližje h^* (da zagotovi učinkovitost). Če bi poznali h^*, bi uporabljali kar samo h^*; A*, ki uporablja h^*, najde optimalno rešitev direktno brez vsakega vračanja [10].

Za zahtevne probleme je težko poiskati dobro hevristično funkcijo. Druck in sod. [27] v uvodu med seboj primerjajo uporabo koncepta dinamičnega programiranja in prioritetno usmerjenega iskanja. Med drugim navajajo, da je običajno glavna ovira pri uporabi algoritmov A* prav nejasnost, kako generirati popolno hevristiko.

4.3.4 Časovna in prostorska zahtevnost A*

Hevristično vodenje pri iskanju najprej najboljši tipično povzroči, da iskanje preišče samo majhen del problemskega prostora (prostora stanj). Na to lahko gledamo kot na zmanjšanje efektivnih skokov med iskanjem (preskokov med različnimi možnostmi). Če b označuje povprečno obličajno število skokov v prostoru stanj in b' označuje povprečno število skokov pri hevristično vodenem iskanju, potem je tipično b' veliko majši kot b.

Kljub takšnemu zmanjšanju preiskanega problemskega prostora je red zahtevnosti algoritma A* še vedno *eksponenten* glede na *globino iskanja*. Ker algoritem

ohranja vsa razvita vozlišča v pomnilniku, to velja tako za prostorsko kot časovno zahtevnost. Katera časovna zahtevnost je bolj kritična, zavisi od praktičnih implementacij na problemih. V splošnem pa je bolj kritična prostorska zahtevnost. Za uporabnika je bolj nerodno, če algoritem v nekaj minutah porabi ves pomnilnik in je nadaljevanje nemogoče; sprejemnljivo pa bi bilo, če bi izvedba algoritma sicer trajala več dni, vendar bi dal rešitev.

Razvitih je bilo več različic algoritma A*, ki optimizirajo porabo prostora na račun časa. Prostorske zahteve se iz eksponentne zahtevnosti zmanjšajo na linearno. Cena za to je ponovno generiranje predhodno že zgeneriranih vozlišč.

Najenostavnejši način za zmanjšanje prostorske zahtevnosti A* je uporaba ideje iterativnega poglabljanja pri hevrističnem iskanju. Rezultat je algoritem A* z iterativnim poglabljanjem (iterative-deepening A* – IDA*). IDA* je praktičen algoritem in enostaven za implementacijo, toda v neugodnih situacijah postane režija pri generiranju vozlišč nesprejemljiva. Boljša, čeprav bolj zapletena tehnika pri varčevanju s prostorom, je uporaba t.i. RBFS (ang. recursive best-first search).

V našem delu bomo uporabili algoritem *rekurzivnega iskanja po načelu najprej najboljši* (ang. Recursive Best-First Search – RBFS [10]), zato ga bomo v naslednjem podpoglavju podrobneje predstavili.

4.3.5 Rekurzivno iskanje po načelu najprej najboljši

RBFS je rekurziven algoritem, ki poskuša posnemati delovanje navadnega usmerjenega iskanja (A*), pri tem pa uporablja samo linearno prostorsko zahevnost. Razlika med A^* in RBFS je ta, da A* hrani v delovnem pomnilniku vsa do sedaj generirana vozlišča, medtem ko RBFS hrani samo vozlišča na trenutni iskalni poti in pa brate in sestre (ang. sibling nodes) teh vozlišč. RBFS ne nadaljuje z iskanjem nepretrgoma po trenutni poti navzdol, ampak se ravna po f-vrednosti najboljše alternativne poti. Ko RBFS začasno preneha s preiskovanjem določenega poddrevesa (ker poddrevo ni več najbolj obetavno), 'pozabi' vozlišča v tem poddrevesu, da prihrani na prostoru.

Prostorska zahtevnost algoritma RBFS je torej linearna glede na globino iskanja (podobno kot pri IDA*). Edini podatek, ki si ga RBFS zapomni o opuščenem poddrevesu, je posodobljena f-vrednost korena poddrevesa z najboljšo f-vrednostjo naslednikov. Na ta način si RBFS zapomni f-vrednost najboljšega lista v pozabljenem poddrevesu in na tej podlagi potem odloči, ali je v nadaljevanju vredno ponovno razvijati to poddrevo. Vrednosti funkcije f se posodobijo pri vračanju rekurzije, podobno kot se to zgodi pri algoritmu A*. Za razlikovanje med 'statičnim' ovrednotenjem funkcije f in posodobljenimi vrednostimi določimo (za vsako vozlišče N) dve funkciji:

- $f(\mathrm{N})$ = vrednost cenilke za vozlišče N. Vrednost je enaka skozi celoten postopek iskanja.
- $F(\mathrm{N})$ = posodobljena f-vrednost za vozlišče N. Vrednost se spreminja te-

kom iskanja, ker je njena vrednost odvisna od razvitih vozlišč-naslednikov vozlišča N.

Funkcija $F(N)$ je določena kot:

- $F(\mathrm{N}) = f(\mathrm{N})$, če vozlišče še ni bilo razvito (ang. expanded) tekom iskanja
- $F(\mathrm{N}) = min\{F(\mathrm{N_i})|\mathrm{N_i}$ je otrok vozlišča N$\}$

Podobno kot A^* tudi RBFS raziskuje poddrevesa v okviru določene *meje* f. Meja je določena z F-vrednostmi bratov in sester vzdolž trenutne iskalne poti (najmanjša F-vrednost brata ali sestre, t.j. F-vrednost najbljižjega tekmeca trenutnemu vozlišču). Predpostavimo, da je N trenutno najbolj obetavno vozlišče v iskalnem drevesu (ima najmanjšo F-vrednost). Vozlišče N se razvije in njegovi otroci se raziščejo do meje *Bound*. Ko je meja presežena (F(N) > Bound), se vsa generirana vozlišča pod N 'pozabijo'. Vendar se popravljena vrednost F(N) ohrani in se uporabi pri odločitvi, kako nadaljevati iskanje.

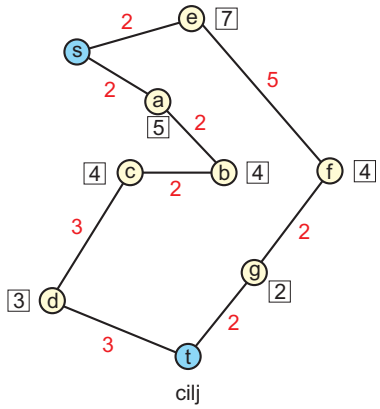
F-vrednost se ne določa samo pri sestopanju, ampak se lahko tudi *deduje* od starša vozlišča. Scenarij se pojavi v primeru, ko imamo vozlišče N, ki ga bomo razvili z iskanjem. Če je F(N) > f(N) potem vemo, da je bilo vozlišče N že enkrat prej razvito in je bila vrednost F(N) določena s strani njegovih otrok, ki so bili odstranjeni iz pomnilnika. Predpostavimo, da ponovno generiramo otrok vozlišča N - vozlišče $\mathrm{N_i}$ in se ponovno izračuna njegova statična vrednost $f(\mathrm{N_i})$. $F(\mathrm{N_i})$ se določi kot:

$$F(\mathrm{N_i}) = max\{F(\mathrm{N}), f(\mathrm{N_i})\})$$

To pomeni, da se v primeru, ko je $f(\mathrm{N_i}) < F(\mathrm{N})$, F-vrednost za vozlišče $\mathrm{N_i}$ podeduje od starša N. Razlog za to početje: ko je bilo vozlišče $\mathrm{N_i}$ pred časom že generirano (in tudi že odstranjeno), je bila vrednost $F(\mathrm{N_i})$ nujno $\geq F(\mathrm{N})$. V nasprotnem primeru bi bila vrednost $F(\mathrm{N})$ manjša.

Delovanje algoritma RBFS bomo pokazali na primeru iskanja najkrajše poti od s do t na preprostem zemljevidu, ki je prikazan na sliki 4.7. Slika 4.8 prikazuje izbrane posnetke stanja razvitih vozlišč na trenutni poti (vključno z brati), ki so hranjena v pomnilniku. V našem primeru sta alternativni poti samo dve. Iskanje nenehno preklaplja med alternativnima potema (podobno kot A*). Ko se preklop med potmi zgodi, se zaradi varčevanja s prostorom prejšnja pot odstrani iz pomnilnika. Rdeče številke poleg vozlišč na sliki 4.8 predstavljajo vrednosti funkcije F za vozlišče.

Na prvem posnetku stanja (A) je vozlišče a najboljše kandidatno vozlišče ($F(a) < F(e)$). Zato se razišče poddrevo pod vozliščem a z mejo $Bound = 9$ ($F(e) = 9$, to je najbližji in v našem primeru tudi edini tekmec). Ko algoritem pride do vozlišča c (posnetek stanja B), se zgodi $F(c) = 10 > Bound$. Pot se začasno opusti, vozlišči c in b se odstranita iz pomnilnika, vrednost $F(c) = 10$ se prenese v vozlišče a, torej $F(a)$ postane 10 (posnetek stanja C). Sedaj e postane najboljši tekmec ($F(e) = 9 <$



Slika 4.7: Zemljevid z dolžinami povezav med mesti. Števila v kvadratih so zračne razdalje do t. Povzeto po [10].

$10 = F(a)$), torej se razišče njegovo poddrevo z mejo $Bound = 10 = F(a)$. Iskanje se ustavi v vozlišču f, ker je $F(f) = 11 > Bound$ (posnetek stanja D). Vozlišče f se odstrani in $F(e)$ postane 11 (posnetek stanja E). Iskanje se spet prenese na vozlišče a z mejo $Bound = 11$. Ko se vozlišče b ponovno generira, ugotovimo, da $f(b) = 8 < F(a)$ zato b podeduje njegovo vrednost F od vozlišča a ($F(b)$ postane 10). Nato se ponovno generira še vozlišče c in c spet podeduje vrednost F od b, torej $F(c)$ postane 10. Iz posnetka stanja (F) je razvidno, da je meja 11 presežena pri vozlišču d, ko je $F(d) = 12$, d, c in b se odstranijo. $F(d) = 12$ se prenese v vozlišče a (posnetek stanja G). Iskanje se ponovno prenese na vozlišče e, kjer dosežemo vozlišče t (posnetek stanja H).

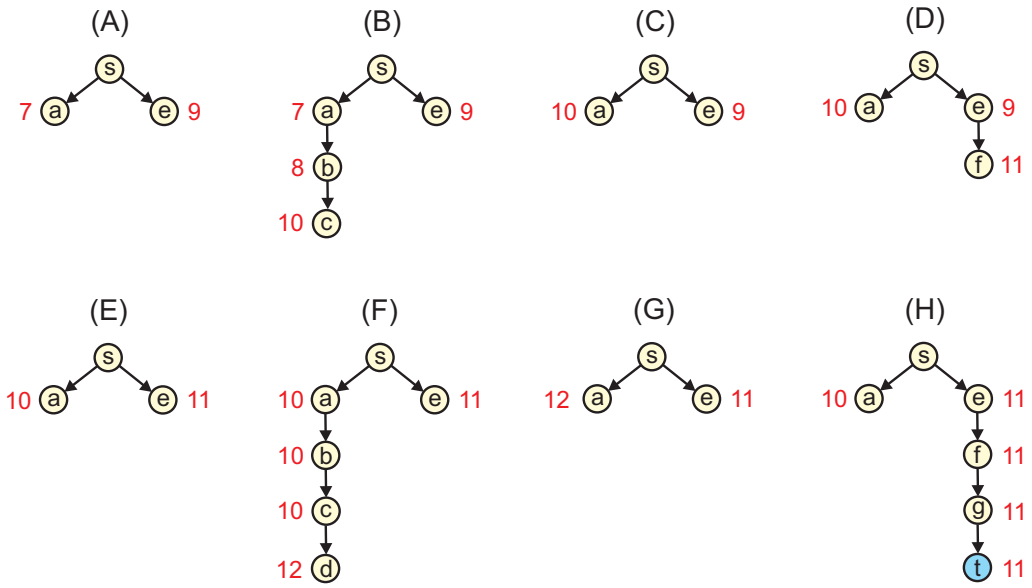
Prostorska zahtevnost algoritma RBFS je linearna glede na globino iskanja v drevesu $O(b \cdot d)$. Cena, ki ga plačamo za to, je režija v obliki časa, potrebnega za ponovno generiranje nekoč že zgeneriranih vozlišč. Časovno kompleksnost je zato precej težko oceniti; odvisna je od točnosti hevristične funkcije in od števila zamenjav med potmi, ko razvijamo vozlišča. Kljub temu pa je režija precej manjša kot pri IDA*. RBFS ima isti najslabši možni scenarij kot IDA*, ko razvije kvadratno število vozlišč, ki jih razvije iskanje po principu best-first search.

Podobno kot A* in za razliko od IDA*, RBFS razvija vozlišča po načelu najprej najboljši tudi, če funkcija f ni monotona. Tako kot A*, je tudi RBFS optimalni algoritem iskanja, če je hevristična funkcija $h(n)$ popolna. RBFS precej slabo izkorišča pomnilnik, ki je na voljo; uporablja samo $O(b \cdot d)$ pomnilnika [81].

4.3.6 Povezava področij

Pri razvoju postopkov razpletanja smo uporabili dva pristopa. Prvi pristop temelji na verjetnosti prehodov med sosednimi stranmi v prepleteni seji. Verjetnosti prehodov med stranmi dobimo iz naučenega markovskega modela prvega reda.

V drugem pristopu smo povezali preiskovanje prostora stanj povezali z marko-



Slika 4.8: Sled algoritma RBFS za problem na sliki 4.7. Številke poleg vozlišč so F-vrednosti vozlišč, ki se spreminjajo med iskanjem. Povzeto po [10].

vskim modelom. Stanja predstavljajo delne razplete prepletene seje, povezave med njimi pa dodeljevanja strani iz prepleta v enega izmed razpletov. Cena povezave med dvemi stanji je določena z verjetnostjo prehoda med dvemi stranmi znotraj razpleta, ki prvo stanje pretvori v drugo. Verjetnost prehodov dobimo iz istega markovskega modela prvega reda, kot pri prvem pristopu. Iskanje rešitvene poti od stanja, ki predstavlja neprepletene sejo, do stanja, ki predstavlja razplet z največjo verjetnostjo, je torej vodeno s pomočjo markovskega modela.

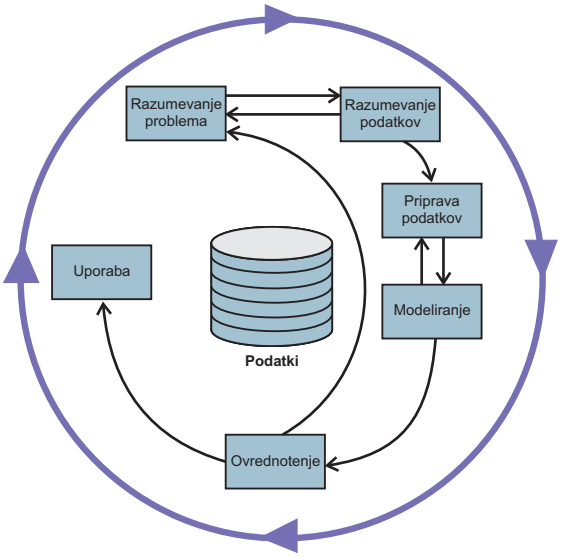
4.4 Zasnova postopkov za razpletanje

4.4.1 Testna metodologija

Pri razvoju modela in postopkov za razpletanje sej smo najprej izbrali metodologijo, po kateri smo delali. Metodologija nam omogoča hitrejši, enostavnejši in bolj zanesljiv razvoj postopkov. Delo je bolje hitrejše, dokumentirano in ponovljivo v kar najkrajšem času. Razvoj postopkov razpletanja je iterativen in ponavljajoč proces. Vrtimo se v krogu priprave podatkov, implementacije postopkov, njihove aplikacije nad pripravljenimi podatki in ovrednotenja rezultatov. Razvoj postopkov razpletanja je zelo podoben projektu odkrivanja zakonitosti v podatkih (ang. data mining project). Za lažje modeliranje in upravljanje projektov odkrivanja zakonitosti v podatkih se uporablja model CRISP-DM [14, 103]. CRISP zagotavlja pregled nad življenjskim ciklom projekta odkrivanja zakonitosti v podatkih. Vsebuje vse faze projekta, naloge in povezave med njimi. Model CRISP-DM smo pri razvoju

uporabili tudi mi. Testna metodologija uporablja ogrodje metodologije CRISP, ki ga bomo predstavili

Življenjski cikel projekta je sestavljen iz šestih faz. Zaporedje faz ne sledi vedno strogo zaporedno. Prehajanje naprej in nazaj med posameznimi fazami je vedno možno in včasih tudi nujno. V odvisnosti od rezultatov trenutne faze se določi, katero fazo bomo izvedli v nadaljevanju. Na sliki 4.9 puščice prikazujejo najpomembnejše in najbolj pogoste odvisnosti med fazami.



Slika 4.9: Faze procesnega modela CRISP-DM.

Zunanji krog simbolizira ciklično naravo postopkov odkrivanja zakonitosti v podatkih. Proces odkrivanja zakonitosti v podatkih se nadaljuje tudi, ko je bila rešitev že nameščena. Informacije, pridobljene na podlagi obstoječih postopkov, lahko pripomorejo k boljšemu poznavanju problema in bolj osredotočenemu reševanju. Ponavljajoči se zaključeni procesi imajo korist od prejšnjih iteracij. Podobno velja tudi za proces razvoja postopkov razpletanja sej. Najprej moramo dodobra spoznati problemsko domeno, da lahko razvijemo učinkovite algoritme. Problemsko domeno spoznavamo skozi posamezne iteracije razpletanja. Na podlagi rezultatov vidimo, kaj je bilo pri iteraciji dobro in kaj je potrebno nadaljnih izboljšav. Ravno tako vidimo ali so popravki parametrov iz prejšnje iteracije prinesli željene rezultate ali ne. CRISP-DM ima sedem glavnih faz:

Razumevanje problema (ang. Business understanding) – Ta faza se nanaša na razumevanje ciljev in zahtev projekta s poslovnega vidika. Znanje o problemski domeni je potrebno preoblikovati v definicije in zahteve orodja, s katerim rešujemo problem. Narediti je potrebno načrt, kako doseči zahteve projekta.

Razumevanje podatkov (ang. Data understanding) – Faza se začne z začetno pridobitvijo samih podatkov in se nadaljuje z aktivnostmi čim boljšega spozna-

vanja podatkov, identifikacije težav s kakovostjo podatkov, prvimi vpogledi v podatke in določanjem zanimivih podmnožic podatkov.

Priprava podatkov (ang. Data preparation) – Vključuje vse aktivnosti za izdelavo končne množice podatkov iz surovih (neprečiščenih) podatkov. Nad končno množico podatkov v naslednji fazi izvedemo razvite postopke in apliciramo modelirna orodja. Faza priprave končnih podatkov se običajno večkrat ponovi v različnih vrstnih redih. Priprava podatkov vključuje izbiro tabel, zapisov, posameznih atributov ter preoblikovanje in čiščenje podatkov za uporabo v modelirnih orodjih.

Modeliranje (ang. Modelling) – V tej fazi apliciramo nad podatki postopke in modele, ki smo jih razvili. Parametre postopkov je potrebno nastaviti na optimalne vrednosti. Različni postopki imajo lahko specifične zahteve za format podatkov. To še zlasti pride do izraza pri razvoju novih postopkov, ko si moramo pripraviti podatke na različne načine v skladu z veljavno programsko rešitvijo v tekoči iteraciji. Vračanje v fazo priprave podatkov je zelo pogosto.

Ovrednotenje (ang. Evaluation) – V tej fazi projekta je kakovosten model (postopki) z vidika zagotavljanja kakovostnih izhodov zgrajen. Pred nadaljevanjem v uporabo je potrebno izvesti še zelo pomembno fazo ovrednotenja modela. Še enkrat je tudi potrebno pregledati vse korake pri izgradnji. Le tako smo lahko prepričani, da pravilno izpolnjuje začetne (poslovne) zahteve. Ključna točka je najti morebitne poslovne zahteve, ki niso dovolj dobro izpolnjene. Na koncu te faze je potrebno določiti, ali so rezultati postopkov dovolj dobri za uporabo.

Uporaba (ang. Deployment) – Izdelava postopkov ali modela v splošnem ne pomeni zaključka projekta. Rezultati morajo biti bodisi razumljivi uporabniku ali pa jih mora biti sposoben vključiti v neko večjo zaključeno celoto. Glede na zahteve lahko v fazo uporabe sodijo samo aktivnosti izdelave nekega poročila ali pa implementacija ponavljajočega se procesa. Postopek uporabe ponavadi izvaja končni uporabnik; kljub temu pa moramo razumeti, kakšni koraki bodo potrebni in v kakšnem okolju bo rešitev delovala ter ustrezno prilagoditi model.

pod pojmom uporaba

4.4.2 CRISP-DM v našem problemu

Pri načrtovanju in razvoju postopkov za razpletanje smo sledili metodologiji CRISP-DM. Uporaba metodologije omogoča boljšo preglednost nad postopki in daje kot rezultat večjo kakovost kode programske opreme. Za vsako fazo smo določili potrebne postopke in izvedli prilagoditve, kjer je bilo potrebno. V nadaljevanju se bomo posvetili posameznim fazam razvoja postopkov za razpletanje sej.

4.4.3 Razumevanje problematike

V prvi fazi smo opredelili problem, ki ga je bilo potrebno rešiti in določili cilje. V postopku predprocesiranja podatkov o klikotoku za uvoz v podatkovno skladišče smo naleteli na problem prepletenih sej, ki negativno vplivajo na kakovost podatkov. Takšnih sej ni možno ločiti na preprost način. Naša naloga je bila razviti postopek za razpletanje takšnih sej. Za rešitev problema lahko uporabimo različne mehanizme. Podrobno smo preučili dosedanje delo na področju klikotoka in morebitne obstoječe pristope za rešitev podobnih problemskih situacij. Pri reševanju problemov s področja klikotoka so se kot primerni izkazali stohastični modeli. Za rešitev problema smo se odločili uporabiti markovske modele in mehanizem preiskovanja prostora stanj.

Namen celotnega procesa je bil izdelati postopke, ki čimbolj verno razpletajo prepletene seje. Pri tem smo se naslonili na podatke o preteklem obnašanju uporabnikov. Razviti postopki razpletanja temeljijo na načelu verjetnosti določenega zaporedja strani. Prehodi med nekaterimi stranmi so bolj verjetni kot med drugimi. Verjetnost prehoda med stranmi je določena s preteklimi prehajanji uporabnikov med stranmi.

4.4.4 Razumevanje podatkov

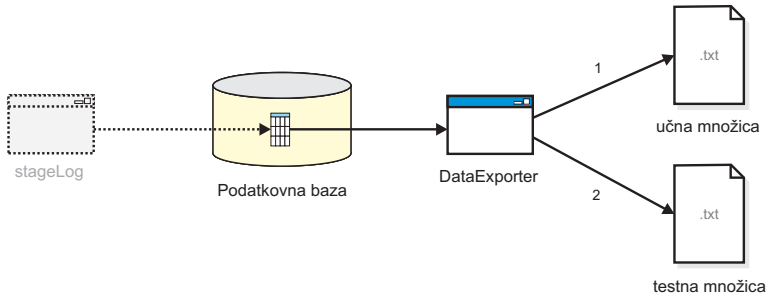
Postopke razpletanja smo testirali nad tremi množicami podatkov: podatki sistema e-Študent, spletne trgovine in umetno generiranimi na podlagi sintetičnega primera. Podatki so med seboj zelo različni. S tem smo se hoteli izogniti primeru, da bi se razvite metode preveč prilagodile na določeno vrsto podatkov. Podatki iz dnevniških datotek spletnih strežnikov so zelo obsežni. Vsebujejo veliko število zapisov, ki zakrivajo celotno sliko in so s stališča pregledovanja in razumevanja podatkov precej moteči. Sem spadajo dostopi do slik, animacij in ostalih pomožnih elementov spletne strani. Ti elementi nudijo malo informacije, kljub temu pa jih ne smemo zavreči do končanega postopka osejevanja. Vira podatkov (t.j. tekstovne dnevniške datoteke) ne moremo neposredno pregledovati z namenom boljšega razumevanja podatkov. Dnevniške datoteke so običajno prevelike za učinkovito pregledovanje v tekstovnem urejevalniku, količina raznovrstnih elementov je prevelika za vpogled v dogajanje. Spletni strežnik zapisuje zahteve kronološko, kot opravlja strežbo zahtevanih virov na strežniku. Zaporedni zapisi v datoteki ne pripadajo nujno zahtevi določenega odjemalca. V danem trenutku lahko poteka strežba več odjemalcem. Zahtevane spletne strani in elementi na teh spletnih straneh se izmenično pošiljajo odjemalcem. Zapisi v dnevniški datoteki zrcalijo vrstni red strežbe.

Za posamezne vire podatkov smo pridobili uporabljeni format zapisa v dnevniško datoteko. Sami viri podatkov so podrobneje predstavljeni v poglavju 5. V jeziku C# smo izdelali aplikacijo za zajem, čiščenje, osejevanje podatkov o klikotoku in zapis v podatkovno bazo. V podatkovni bazi, kjer smo dobili prečiščene in osejene podatke, je možen neprimerno boljši vpogled v podatke. Spoznavanje s podatki je terjalo veliko časa. Glavni razlog za to je zelo nestrukturiran vir podatkov.

Spoznati je bilo potrebno lastnosti uporabniških sej, pravilno določiti povprečen čas uporabniške seje, identificirati zapise robotov spletnih iskalnikov in se soočiti s posebnostmi posameznega vira. Največ težav smo imeli z učinkovitim ostranjevanjem zapisov spletnih robotov. Odstranjevanje njihovih zapisov je ključno za kakovost podatkov, saj njihove seje zelo kvarno vplivajo na rezultate analiz. Pri sistemu e-Študent je bilo potrebno prilagoditi postopek čiščenja podatkov zaradi dodatnih zapisov, dodanih s strani aplikacije in nekaterih manjkajočih podatkov. Pri spletni trgovini je bilo potrebno paziti na strežbo oglasnih pasic na drugih spletnih mestih. Aplikacijo (logParser) smo spreminjali in nadgrajevali v več iteracijah v skladu s poznavanjem podatkov. Poznavanje realnih podatkov iz obeh virov nam je omogočilo izdelavo preprostega testnega modela spletnega mesta, na podlagi katerega smo potem lahko generirali umetne podatke.

4.4.5 Priprava podatkov

Postopek razpletanja sej zahteva vhodne podatke v točno določenem formatu. Potrebujemo učno in testno množico podatkov. Učna množica podatkov služi za učenje markovskega modela, testna množica pa predstavlja prepletene seje, ki se v postopku razpletanja ločijo na sestavne dele. Učna množica podatkov je sestavljena iz čistih (neprepletenih) sej. Testna množica pa vsebuje prepletene seje s podatki o njihovih sestavnih delih. Podatki s čistimi sejami se nahajajo v podatkovni bazi v tabeli očiščenih uporabniških sej kot izhod programa *stageLog* na sliki 4.10. Sivo obarvan del je podrobneje opisan v poglavju 5. Vhod postopka za razpletanje ni neposredno tabela očiščenih uporabniških sej. Razlog je v drugačni strukturi tabele za sistem e-Študent in spletno trgovino. V opisu podatkov smo namreč videli, da imata oba vira na voljo drugačen nabor atributov v dnevniški datoteki. Določili smo format za učno in testno množico podatkov, ki je enak ne glede na vir podatkov, za umetno generirane in realne podatke. Izdelali smo aplikacijo (DataExporter), ki v skladu s parametri iz čistih sej v tabeli podatkovne baze izdela ustrezno množico učnih in testnih podatkov in jih zapiše v datoteko. Postopek priprave podatkov za postopek razpletanja prikazuje slika 4.10.



Slika 4.10: Postopek priprave podatkov za proces razpletanja sej.

Čiste seje v tabeli podatkovne baze se v razdelijo na dva dela. Določen odsto-

tek zapisov se uporabi za generiranje učne množice (tipično 70%), preostali del pa predstavlja osnovo za izdelavo testne množice. Množica učnih podatkov vsebuje klikotok s čistimi, nepreplетenimi sejami. Format zapisa učnih podatkov je enak formatu umetno generiranega klikotoka v tabeli 5.1. Namenjena je za učenje markovskega modela, ki se uporablja v fazi razpletanja. Faza priprave učnih podatkov je preprosta. Čiste seje se že nahajajo v podatkovni bazi. Potrebno je določiti najmanjšo in največjo dovoljeno dolžino seje, narediti naključen vzorec in jih zapisati v pravilnem formatu na izhod.

V fazi razvoja postopka razpletanja sej smo uporabili za testno množico prepletene seje, ki smo jih generirali iz čistih sej. Pri prepletanju smo uporabili znanje pridobljeno o dejanskih prepletenih sejah. Generirane preplete smo opremili z metapodatki – to so podatki o sejah, ki sestavljajo prepleteno sejo. Metapodatki so nam omogočili primerjavo rezultatov razpleta z dejanskimi sestavnimi sejami. Ovrednotenje pravilnosti razpleta brez metapodatkov o prepletenih sejah bi bilo težko. Primer prepletene seje iz množice testnih podatkov prikazuje slika 4.11.

Prvi dve vrstici prikazujeta elementarni (čisti) seji prepleta, nato pa sledi klikotok prepletene seje. Vsaka elementarna seja je navedena v obliki opombe v svoji vrstici. Predstavljena je kot zaporedje imen strani, ločenih med seboj s presledki. Prepletena seja na sliki je sestavljena iz dveh čistih sej. Vidimo, da je format klikotoka podoben tistemu iz učne množice. Razlika je le v imenu atributa *uporabnik*. Za potrebe razvoja postopka razpletanja ostali atributi (odjemalec, prejšnja stran), ki so sicer na voljo v podatkovni bazi, niso potrebni. Vsak dostop prepletene seje je opisan s podatkom o uporabniku, datumu, času in zahtevani spletni strani. Atribut *uporabnik* združuje podatek o zaporedni številki prepleta in čiste seje znotraj prepleta. Na sliki 4.11 vidimo, da gre za tretji preplet, kjer se menjujeta prva in druga uporabniška seja. Dejanska imena spletnih strani so včasih zelo nerodna za rokovanje. Dolžina imen dejanskih strani je zelo različna, predvsem daljša so manj fleksibilna za predstavitev. Za potrebe testiranja smo imena dejanskih spletnih strani zamenjali s krajšimi. Stran je predstavljena s kombinacijo *Sx*, kjer *x* predstavlja zaporedno številko strani znotraj spletnega mesta.

Postopke razpletanja sej smo preizkusili na testni množici prepletenih sej, ki smo jo zgenerirali na različne načine, da smo lahko primerjali uspešnost razpletanja posameznih vrst prepletenih sej. Generirane preplete smo primerjali z dejanskimi prepletenimi sejami in določili parametre generatorja, ki generira prepletene seje čimbolj podobne dejanskim. Pri generiranju prepletenih sej smo upoštevali verjetnost prehodov med podsejami, različne podmnožice čistih sej in velikost gradnika prepleta. Verjetnost prehoda med podsejami p_p določa, kako velika verjetnost je, da bo naslednja stran v prepleteni seji pripadala drugi čisti seji. Manjša kot je p_p, manj je prehodov med podsejami v prepleteni seji, več strani zaporedoma pripada isti čisti seji. Večja kot je verjetnost prehoda p_p, več je prehodov med podsejami (uporabnik več preklaplja med okni).

Preverjali smo tudi rezultate razpletanja sej, ki so sestavljene iz določene podmnožice čistih sej. Primer: pri spletni trgovini imamo dve vrsti uporabnikov, ano-

```
// S0 S3 S1 S7 S17
// S10 S4 S11 S19
[3_drg] 2007.04.17 07:29:09 S10
[3_drg] 2007.04.17 07:29:13  S4
[3_drg] 2007.04.17 07:29:17 S11
[3_prv] 2007.04.17 07:29:25  S0
[3_drg] 2007.04.17 07:29:33 S19
[3_prv] 2007.04.17 07:29:42  S3
[3_prv] 2007.04.17 07:29:42  S1
[3_prv] 2007.04.17 07:29:42  S7
[3_prv] 2007.04.17 07:29:42 S17
```

Slika 4.11: Primer prepletene seje iz testne množice, uporabljene v fazi razvoja postopka.

nimne in prijavljene, ki so opravili nakup. Prepletene seje smo generirali iz čistih sej:

- uporabnikov, ki so se prijavili,
- vseh uporabnikov.

Pri gradnji prepletene seje lahko prepletamo posamezne strani čistih sej ali pa segmente čistih sej. Vsak segment je tipično sestavljen iz več strani, najmanjša in največja dolžina segmenta je podana s parametroma. Pri uporabi segmentov kot zrna smo poskusili generirati prepletene seje, ki čimbolj sovpadajo z načinom dela uporabnikov. Uporabnik tipično v prvem oknu brskalnika opravi neko zaključeno nalogo, potem pa preklopi na drugo okno brskalnika, kjer opravi neko drugo nalogo. V času trajanja interakcije s spletnim strežnikom tako preklaplja med okni brskalnika. Prehodi med segmenti različnih sej so lahko obvezni, lahko pa podobno kot zgoraj uporabimo parameter verjetnosti prehoda med odsejami p_p. V tem primeru si lahko v prepleteni seji sledita dva ali več segmentov iz iste čiste seje. Če zrno pri prepletanju tvori ena stran, potem se za vsako pozicijo i v prepleteni seji naključno izbere naslednja stran ene izmed čistih sej. Predstavljamo si lahko prepletanje segmentov s segmentom dolžine 1.

Pri generiranju prepletenih sej se določa tudi nivo različnosti prepletenih sej. Posamezna čista seja se lahko uporabi za generiranje izključno enega samega prepleta, lahko pa za generiranje enega ali več prepletov.

4.4.6 Modeliranje

V tej fazi smo preizkušali različne metode razpletanja sej nad podatki, ki smo jih pripravili v predhodnih postopkih. Postopke za razpletanje prepletenih sej in vse potrebne komponente za pravilno delovanje smo implementirali v programskem jeziku

java. Razpletanje sej temelji na vzorcih preteklih obnašanj uporabnikov. Za modeliranje preteklega obnašanja uporabnikov smo uporabili markovski model. Implementirali smo ogrodje, ki omogoča kreiranje markovskih modelov različnih redov ter ga lahko enostavno vključimo in uporabimo v različnih postopkih razpletanja. Predhodno pripravljena učna množica podatkov se uporabi za učenje markovskih modelov.

Izdelali in preizkušali smo različne metode razpletanja in z njimi, ob pomoči naučenih markovskih modelov, izvedli razpletanje sej. Rezultate smo s pomočjo metapodatkov o vsebovanih sejah ovrednotili in jih predstavili z grafom. Poleg samih končnih rezultatov so nas bolj zanimali razlogi, ki so privedli do konkretnega končnega rezultata. Preverjanje posamičnih primerov na podatkih iz dejanskih spletnih mest je težje zaradi obsežnosti podatkov. Za ta namen smo uporabili testno množico umetno generiranih podatkov, kjer smo za napačno razpletene seje pogledali, kaj je verjeten razlog za napako. Isto metodo smo načeloma izvedli večkrat z drugačnim naborom parametrov in drugače prepletenimi sejami. Pri razvijanju metode smo pazili, da metoda dovolj dobro dela za vse vire podatkov.

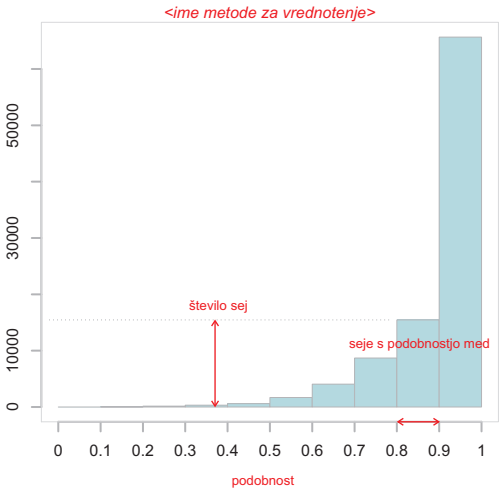
Poleg preizkušanja metod razpletanja smo v tej fazi izdelali tudi vrsto postopkov za testiranje pravilnosti delovanja in razumevanja podatkov. Narava podatkov o klikotoku je takšna, da je iz samih podatkov težko videti celotno sliko. Naredili smo aplikacijo za analizo prepletenih sej v sistemu e-Študent in ugotovili značilnosti dejanskih prepletenih sej. Ugotovitve smo uporabili pri postopku kreiranja testnih prepletenih sej. Generirane prepletene seje smo grafično predstavili, s pregledom analizirali in ugotavljali pravilnost delovanja postopkov. Posebno pozornost smo namenili ugotavljanju začetnih stanj prepletenih sej. V tem delu smo se posvetili analizi pravilnosti delovanja postopkov in nastavitvam parametrov postopkov.

4.4.7 Ovrednotenje

Uspešnost metode pri razpletanju smo ugotovili z vrednotenjem rezultatov razpleta. Primerjali smo dejanske vrednosti razpletenih sej in sestavne čiste seje, ki smo jih uporabili pri kreiranju prepletenih sej. Metapodatke o sestavnih sejah smo imeli za umetno generirane prepletene seje. Uporabili smo več metod vrednotenja, ki so podrobneje predstavljene v razdelki 4.5. Za vsako razpletanje smo narisali histogram števila sej glede na podobnost dejanskim sejam. Slika 4.12 prikazuje primer histograma in pomen podatkov na njem. Os x prikazuje podobnost med razpletenimi sejami in čistimi sejami, ki so bile uporabljene za generiranje prepleta. Vrednosti bližje 1 pomenijo bolj podobne seje in posledično boljši rezultat. Os y prikazuje število sej, ki je padlo v ta interval.

4.4.8 Uporaba

V fazi uporabe smo razvite postopke za razpletanje sej vključili v postopek ETL, ki skrbi za zajem, čiščenje, preoblikovanje in nalaganje podatkov o klikotoku v spletno



Slika 4.12: Graf za prikaz rezultatov razpletanja sej.

podatkovno skladišče.

4.5 Ovrednotenje procesa razpletanja

Razpletene seje je potrebno ovrednotiti. Ugotoviti moramo, kako dobro so bile v postopku razpletanja razdeljene na sestavne dele. To naredimo tako, da jih po nekem postopku primerjamo s sejami, ki bi jih morali dobiti kot rezultat. Postopek ovrednotenja po ključu ovrednoti podobnost sej in vrne merilo podobnosti med sejama.

Vsaka seja je sestavljena iz zaporedja ogledanih spletnih strani. Na uporabniško sejo lahko torej gledamo kot na zaporedje (ang. sequence). Ovrednotenje uspešnosti razpletanja lahko prevedemo na problem ocenjevanja podobnosti zaporedij simbolov. V našem primeru vsak simbol predstavlja določena spletna stran. Primerjanje zaporedja simbolov med seboj in določanje podobnosti med seboj je ena izmed osnovnih nalog strojnega prevajanja (ang. machine translation) in računske biologije (ang. computational biology) [48]. V svoji osnovi imamo dve zaporedji simbolov med seboj za bolj podobni, čimveč simbolov imata skupnih in čimbolj je vrstni red simbolov med njima podoben. Obstaja veliko metod za merjenje podobnosti med dvema zaporedjema. V literaturi je bilo predstavljenih več metod za vrednotenje podobnosti, ki temeljijo na popolnem ujemanju, operacijah urejanja zaporedja (ang. edit-distance), najdaljšem skupnem podzaporedju in sovpadanju n-teric (ang. n-gram) elementov (ang. n-gram co-occurrence), itn [52]. Za ovrednotenje uspeha razpletanja sej smo se odločili uporabiti pet postopkov, ki temeljijo na naštetih štirih metodah. Dodali smo še izpeljanko metode najdaljšega skupnega podzaporedja, metodo uteženega najdaljšega skupnega podzaporedja.

4.5.1 Popolno ujemanje

Metoda popolnega ujemanja temelji na 100% enakosti dveh zaporedij. Poimenujmo zaporedje, ki ga hočemo ovrednostiti kot kandidatno zaporedje; ciljno zaporedje pa imenujmo referenčno zaporedje. Rezultat ovrednotenja po tej metodi je lahko samo uspešno (1) ali neuspešno (0). Če imamo kandidatno zaporedje $Y = [y_1, y_2, \ldots, y_n]$ in referenčno zaporedje $X = [x_1, x_2, \ldots, x_m]$, potem metoda vrne 1 takrat, ko je $n = m$ in ko za vsak i velja: $x_i = y_i$. Če to ne velja, se zaporedji obravnavata kot različni. Vrednotenje po tej metodi je zelo striktno in preimerno za vrednotenje takrat, ko nas zanimajo samo najbolj kakovostni podatki. Po drugi strani pa so po tej metodi enako slaba zaporedja, ki so med sabo popolnoma različna in zaporedja, ki imajo med sabo zamenjana samo znaka na mestu i in j.

4.5.2 Razdalja med dvema zaporedjema

Drugi pristop merjenja podobnosti med zaporedjema temelji na *merjenju razdalje med zaporedjema* (ang. distance measures). Pogosto pri pristopu merjenja razdalje med zaporedji določimo množico *operacij urejanja*, kot je vstavljanje ali brisanje simbola, vključno s ceno posamezne operacije. Razdalja med dvema zaporedjema (ang. edit-distance) je določena kot vsota cen najcenejše verige operacij, ki prvo zaporedje pretvorijo v drugega. Če množico vseh operacij, ki so na voljo, sestavljajo samo tri operacije vstavljanja, brisanja in zamenjave, cena vsake operacije pa je 1, dobimo *Levenshteinovo razdaljo* (ang. Levenshtein distance) [51]. Posplošitev Leveshteinove razdalje je Damerau-Levenshteinova razdalja, kjer imamo tudi operacijo transpozicije dveh sosednjih elementov [22]. Primer, Levenshteinova razdalja med besedama 'miza' in 'možak' je enaka 3, ker so potrebne 3 operacije urejanja, ki pretvorijo prvi niz v drugega; z manj operacijami ni možno narediti pretvorbe:

- miza → moza (zamenjava 'i' z 'o')
- moza → moža (zamenjava 'z' z 'ž'),
- moža → možak (vstavljanje 'k' na koncu).

Klasične mere na osnovi razdalj med zaporedjema se ne skladajo dobro z razmislekom, da sta dve zaporedji podobni, če zamenja pozicijo cel blok simbolov. Predpostavimo, da so A, B, C, D bloki simbolov. Predpostavimo tudi, da bloka B in C nimata skupnih simbolov. Da bi pretvorili zaporedje $ABCD$ v $ACBD$ z uporabo Levenshteinovih operacij, moramo pobrisati vse simbole iz B in jih vstaviti za C (ali obratno), kar pomeni $2 \cdot min\{|B|, |C|\}$. Vidimo lahko, da je kazen za premikanje bloka enaka konstanti – dolžini bloka. Leusch in sod. [48] sicer v svojem delu predlagajo vpeljave nove operacije premikanja bloka. Kljub temu smo se za vrednotenje razpletanja odločili uporabiti klasično Levenshteinovo razdaljo.

4.5.3 Najdaljše skupno podzaporedje

Različice Levenshteinove razdalje lahko dobimo s spreminjanjem množice operacij, ki so na voljo. Tak primer je najdaljše skupno podzaporedje (ang. longest common subsequence). Zaporedje $Z = [z_1, z_2, ..., z_n]$ je podzaporedje nekega drugega zaporedja $X = [x_1, x_2, ...x_m]$, če obstaja strogo naraščajoče zaporedje indeksov $i_1, i_2, ...i_k$ v zaporedju X tako da za vse $j = 1, 2, ..., k$ velja $x_{i_j} = z_j$ [21]. Če imamo dve zaporedji X in Y, je najdaljše skupno podzaporedje (LCS) X in Y skupno podzaporedje z največjo dolžino. Daljše kot je najdaljše skupno podzaporedje dveh zaporedij, bolj sta si podobni. LCS dveh zaporedij dolžine m in n lahko s pomočjo tehnike dinamičnega programiranja izračunamo v času $O(mn)$.

Prednost LCS je, da ne zahteva zaporednih ujemanj simbolov, pač pa samo ujemanje na nivoju zaporedja (ne zahteva strogo zaporednih istoležnih simbolov). Rezultat si lahko predstavljamo kot *n-terico* elementov na nivoju zaporedja z upoštevanim vrstnim redom. Z izrazom n-terica označujemo podzaporedje n elementov iz danega zaporedja. Druga prednost je, da samodejno dobimo največjo skupno n-terico na nivoju zaporedja brez vnaprejšnjega določanja dolžine n. LCS pravilno ovrednoti tudi strukturo zaporedja. Povedano pokažimo še na primeru, kjer je prvi stavek X referenčni in ga primerjamo z Y_1 in Y_2:

X : danes je lep dan
Y_1 : danes ni lep dan
Y_2 : lep dan ni danes

LCS oceni zaporedje Y_1 z rezultatom $\frac{3}{4} = 0,75$, Y_2 pa z rezultatom $\frac{2}{4} = 0,5$, kar pomeni da je Y_1 za LCS bolj podoben prvemu zaporedju. Tako vrednotenje je v skladu z našimi željami. Vendar pa LCS upošteva samo glavno najdaljše skupno podzaporedje; druga najdaljša skupna podzaporedja ali krajša podzaporedja niso upoštevana pri rezultatu, vendar bi lahko pomembno vplivala nanj. Vzamemo za primer zaporedje:

Y_3 : lep dan danes je

Če zaporedje Y_3 primerjamo z referenčnim zaporedjem X, LCS upošteva samo najdaljše skupno podzaporedje “lep dan“ ali pa “danes je“, nikakor pa ne obeh hkrati. Rezultat za zaporedje Y_3 je isti kot za zaporedje Y_2, kar gotovo ni najboljše. Očitno je, da je zaporedje Y_3 bolj podobno referenčnemu kot pa Y_2, saj ima samo zamenjan vrstni red dveh blokov. Metoda, ki odpravlja to pomanjkljivost je skip-bigram, ki jo bomo predstavili nadalje.

Lin in sod. [52] so predlagali uporabo *mere F* (ang. F-measure) za ocenjevanje podobnosti dveh zaporedij X dolžine m in Y dolžine n, kjer se X obravnava kot referenčno zaporedje, Y pa kandidatno. Harmonično sredino, t.i. mero F, je prvi predlagal Van Rijsbergen [94]. Za potrebe ocenjevanja kakovosti strojnega prevajanja tekstov je mero F uporabil tudi Turian [93] in pokazal, da se lahko kosa z

drugimi metodami. Mera F za LCS je določena kot:

$$R_{\text{lcs}} = \frac{\text{LCS(X, Y)}}{m} \tag{4.13}$$

$$P_{\text{lcs}} = \frac{\text{LCS(X, Y)}}{n} \tag{4.14}$$

$$F_{\text{lcs}} = \frac{(1+\beta^2)R_{\text{lcs}}P_{\text{lcs}}}{R_{\text{lcs}} + \beta^2 P_{\text{lcs}}} \tag{4.15}$$

$LCS(X,Y)$ predstavlja najdaljše skupno podzaporedje zaporedij X in Y. Za β velja $\beta = P_{lcs}/R_{lcs}$, ko je $\partial F_{lcs}/\partial R_{lcs} = \partial F_{lcs}/\partial P_{lcs}$. Mera F (F_{lcs}) je enaka 1, ko sta X in Y med seboj enaka, ker je v tem primeru $LCS(X,Y) = m$ ali n. $F_{lcs} = 0$, ko je $LCS(X,Y) = 0$ oziroma ni nič skupnega med zaporedjema X in Y. Mera F združuje natančnost (ang. precision) in preklic (ang. recall). V našem primeru sta mera in preklic osnovana na LCS. R_{lcs} predstavlja preklic – delež simbolov v zaporedju X, ki so prisotni tudi v Y. P_{lcs} predstavlja natančnost – delež simbolov v Y, ki jih najdemo tudi v X. S pomočjo faktorja β določamo s kolikšno utežjo se upoštevata natančnost in preklic v meri F. Če je $\beta = 1$, dobimo $F(R,P) = \frac{2RP}{R+P}$ kar pomeni, da sta oba upoštevana enakovredno.

Mero F na podlagi LCS, t.j. enačbo (4.15) so Lin in sod. imenovali ROUGE-L. Mero F na podlagi LCS bomo na podoben način uporabili tudi mi. Zaporedje X bo pri nas predstavljalo pravilno razpleteno, Y pa dejansko razpleteno sejo.

4.5.4 Uteženo najdaljše skupno podzaporedje

Kot smo videli, ima metoda vrednotenja LCS veliko pozitivnih lastnosti. Ima pa tudi pomanjkljivost, da ne zna razločevati med najdaljšimi skupnimi podzaporedji v odvisnosti od pozicij drugih elementov v zaporedju. Najbolje se to vidi na naslednjem primeru, kjer imamo referenčno zaporedje X, in dve zaporedji Y_1 in Y_2, ki ju ocenjujemo:

$$\begin{aligned} X: &\quad [\underline{\text{A}}\ \underline{\text{B}}\ \underline{\text{C}}\ \underline{\text{D}}\ \text{E}\ \text{F}\ \text{G}] \\ Y_1: &\quad [\underline{\text{A}}\ \underline{\text{B}}\ \underline{\text{C}}\ \underline{\text{D}}\ \text{H}\ \text{I}\ \text{K}] \\ Y_2: &\quad [\underline{\text{A}}\ \text{H}\ \underline{\text{B}}\ \text{K}\ \underline{\text{C}}\ \text{I}\ \underline{\text{D}}] \end{aligned}$$

Mera F na podlagi LCS je enaka tako za Y_1 kot za Y_2. Vendar bi bilo v tem primeru zaporedje Y_1 verjetno boljša izbira kot Y_2. V zaporedju Y_1 je največje skupno podzaporedje strnjeno, brez prekinitev. Metodo LCS lahko enostavno izboljšamo tako, da si v postopku dinamičnega programiranja, ki izračunava LCS, zapomnimo dolžino zaporednih ujemanj do sedaj. Vpeljemo dodatno spremenljivko k, ki označuje dolžino zaporednih ujemanj, ki se zaključujejo na mestu X_i in Y_j. Lin in sod. [52] so metodo poimenovali uravnoteženi LCS (Weighted LCS). Če imamo dve

zaporedji X in Y, ovrednotimo oceno po postopku WLCS s pomočjo naslednjega postopka dinamičnega programiranja:

```
for(i = 0; i <= m; i++)
  c[i][j] = 0      // inicializacija tabele za LCS
  w[i][j] = 0      // inicializacija tabele za zaporedno ujemanje
for (i = 0; i <=m; i++)
  for(j =1; j <=n; j++)
    if (xi == yj)
      // dolzina zaporednih ujemanj na mestu i-1 in j-1
      k = w(i-1, j-1)
      c[i][j] = c[i-1][j-1] + f(k+1) - f(k)
      // zapomnimo si dolzino zaporednih ujemanj na mestu i,j
      w[i][j] = k + 1
    else
      if (c[i-1][j] > c[i][j-1])
        c[i][j] = c[i-1][j]
        w[i][j] = 0         // na mestu i,j ni ujemanja
      else
        c[i][j] = c[i][j-1]
        w[i][j] = 0         // na mestu i,j ni ujemanja
WLCS(X,Y) = c[m][n]
```

c je tabela za dinamično programiranje, kjer $c[i][j]$ hrani rezultat WCLS na mestu i zaporedja X in j zaporedja Y. w je tabela, ki hrani zaporedna ujemanja znakov v tabeli c na mestu i in j. f je utežitvena funkcija zaporednih ujemanj v tabeli na mestu $c[i][j]$. Z uporabo različnih utežitvenih funkcij f lahko vplivamo na algoritm WCLS tako, da damo večji pomen zaporednim ujemanjem znakov, ki sestavljajo LCS.

Za utežitveno funkcijo f mora veljati lastnost $f(x+y) > f(x)+f(y)$ za vsa pozitivna števila x in y. To pomeni, da zaporednim ujemanjem znakov CLS pripišemo večji rezultat kot zaporedjem, kjer je med elementi LCS presledek. Primer take funkcije je $f(k) = \alpha k - \beta$ za $k >= 0, \alpha, \beta > 0$. Funkcija dodeli negativen dodatek $-\beta$ k rezultatu za vsak nezaporedni niz elementov. Druga možna skupina funkcij so polinomi oblike k^{α}, kjer je $\alpha > 1$. Zaradi normalizacije končnega rezultata (mere F) so najbolj primerne funkcije, ki imajo elementarno inverzno funkcijo. Primer: $f(k) = k^2$ ima elementarno inverzno funkcijo $f^{-1}(k) = \sqrt[2]{k}$. Mero F, ki temelji na WCLS, lahko izračunamo na način, ki je prikazan v formuli (4.18). Predpostavljamo, da imamo zaporedji X dolžine m in Y dolžine n.

$$R_{\text{wlcs}} = f^{-1}\left(\frac{\text{LCS}(\text{X},\text{Y})}{f(m)}\right) \tag{4.16}$$

$$P_{\text{wlcs}} = f^{-1}\left(\frac{\text{LCS}(\text{X},\text{Y})}{f(n)}\right) \tag{4.17}$$

$$F_{\text{wlcs}} = \frac{(1+\beta^2)R_{\text{wlcs}}P_{\text{wlcs}}}{R_{\text{wlcs}}+\beta^2 P_{\text{wlcs}}} \tag{4.18}$$

Funkcija f^{-1} je inverzna funkcija funkcije f. Lin in sod. [52] so mero F na podlagi postopka WLCS, ki jo predstavlja enačba (4.18), poimenovali ROUGE-W. Pri uporabi enačbe (4.18) in utežitvene funkcije $f(k) = k^2$, dobimo rezultat za zaporedji Y_1 in Y_2 0.571 in 0.286. WLCS metoda torej razvršča zaporedje Y_1 višje kot Y_2. Uporabljena je bila polinomska funkcija oblike k^α. Funkcijo iz te družine smo pri ovrednotenju razpletov sej z metodo ROUGE-W uporabili tudi mi.

4.5.5 Statistika sopojavitev ločenih dvojic elementov

Kot *dvojico elementov* obravnavamo skupino dveh skupaj zapisanih črk, besed, simbolov (ang. bigram). *Ločene dvojice* elementov pa obravnavamo kot katerikoli par besed ali črk v zaporedju, kjer je dopuščena vmesna vrzel (ang. skip-bigram). Statistika sopojavitev ločenih dvojic elementov meri prekrivanje ločenih dvojic elementov med izvornim zaporedjem in ocenjevanim zaporedjem [52]. Za prikaz uporabimo primer iz prejšnjega dela:

$$\begin{aligned} X: &\quad \textit{danes je lep dan} \\ Y_1: &\quad \text{danes ni lep dan} \\ Y_2: &\quad \text{lep dan ni danes} \\ Y_3: &\quad \text{lep dan danes je} \end{aligned} \tag{4.19}$$

Vsak stavek ima $C(4,2) = \binom{4}{2} = 6$ ločenih dvojic elementov (ang. skip-bigrams). Na primer, zaporedje X ima naslednje ločene dvojice elementov: (“danes je”, “danes lep”, “danes dan”, “je lep”, “je dan”, “lep dan”). Zaporedje Y_1 ima tri ločene dvojice elementov, ki so skupne tistim iz X: (“danes lep”, “danes dan”, “lep dan”). Y_2 ima eno ločeno dvojico, ki se ujema z X (“lep dan”), Y_3 pa dve (“lep dan”, “danes je”). Če imamo izvorno zaporedje X dolžine m in ocenjevano zaporedje Y dolžine n, izračunamo mero F na podlagi ločenih dvojic elementov na naslednji način:

$$R_{\text{skip2}} = \frac{\text{SKIP2}(\text{X},\text{Y})}{C(m,2)} \tag{4.20}$$

$$P_{\text{skip2}} = \frac{\text{SKIP2}(\text{X},\text{Y})}{C(n,2)} \tag{4.21}$$

$$F_{\text{skip2}} = \frac{(1+\beta^2)R_{\text{skip2}}P_{\text{skip2}}}{R_{\text{skip2}}+\beta^2 P_{\text{skip2}}} \tag{4.22}$$

$SKIP2(X,Y)$ predstavlja število ločenih dvojic elementov med X in Y, $\beta = P_{skip2}/R_{skip2}$ in velja $\partial F_{skip2}/\partial R_{skip2} = \partial F_{skip2}/\partial P_{skip2}$. $C(m,2)$ in $C(n,2)$ predstavljata število različnih dvojic elementov, ki jih lahko sestavimo iz elementov zaporedja X oz Y. Lin in sod. [52] imenujejo mero F na podlagi ločenih dvojic elementov v enačbi (4.22) ROUGE-S. Uporaba te metode in njeni rezultati so bolj intuitivni kot pri metodi ROUGE-L ali metodi BLEU [67], ki ravno tako temelji na sopojavitvah n-teric elementov v dveh zaporedjih. Prednost metode ROUGE-S pred BLEU je ta, da ne zahteva zaporednih pojavitev besed, kljub temu pa je še vedno občutljiva na vrstni red elementov v zaporedju. Metoda LCS najde samo najdaljše skupno podzaporedje dveh zaporedij, metoda ločenih dvojic elementov pa najde vse ujemajoče pare elementov. Za ocenjevanje kakovosti razpletanja sej bomo uporabili opisano različico. Metodo lahko nadgradimo s pomočjo uporabe parametra d_{skip}, ki nam določa maksimalno razdaljo med dvema elementoma, ki tvorita ločeno dvojico (skip-bigram). V tem primeru združujemo v pare samo tiste elemente v zaporedju, ki so samo d_{skip} elementov narazen.

4.5.6 Interpretacija rezultatov

Poglejmo si primer vrednotenja podobnosti dveh enostavnih zaporedij z zgoraj opisanimi metodami. Primerjajmo dve zaporedji X in Y s petimi elementi.

X : danes je resnično lep dan
Y : danes ni resnično lep dan

Na podlagi metod vrednotenja podobnosti med zaporedji, dobimo rezultate v tabeli 4.1.

Tabela 4.1: Podobnost zaporedij X in Y glede na metode vrednotenja.

popolna enakost	edit-distance	LCS	WLCS	skip-bigram
0	0,8	0,8	0,6	0,6

Metoda vrednotenja na podlagi popolne enakosti pove ali sta dve zaporedji popolnoma enaki. V našem primeru se X in Y razlikujeta v elementu na drugem mestu, zato metoda vrne vrednost 0.

Vrednotenje na podlagi razdalje med zaporedji meri podobnost na podlagi relativne razdalje med zaporedji. Drugo zaporedje Y pretvorimo v prvo X z eno operacijo zamenjave 'ni' v 'je'. Največja možna razdalja med zaporedjema X in Y je 5 (število operacij potrebno za izgradnjo zaporedja iz praznega zaporedja). Relativna razdalja med zaporedjema X in Y je torej 1/5 = 0,2. Iz tega sledi, da je podobnost 1 - 0,2 = 0,8.

Metoda LCS vrne razmerje med dolžino najdaljšega skupnega podzaporedja in dolžino najdaljšega zaporedja. Dolžina najdaljšega skupnega zaporedja X in Y je 4

(danes resnično lep dan), dolžina najdaljšega zaporedje pa je enaka 5. Metoda LCS torej vrne razultat 4/5 = 0,8.

Vrednotenje na podlagi uteženega najdaljšega skupnega zaporedja (WLCS) za razliko od LCS upošteva samo neprekinjeno najdaljše skupno podzaporedje. V našem primeru je to podzaporedje 'resnično lep dan' dolžine 3. Po analogiji z LCS vrne metoda razmerje 3/5 = 0,6.

Statistika sopojavitev ločenih dvojic elementov vrne razmerje med skupnimi sopojavitvami dvojic elementov in vsemi ločenimi dvojicami elementov najdaljšega zaporedja. Vseh ločenih dvojic za zaporedje dolžine 5 je enako $C(5,2) = \binom{5}{2} = 10$. Za obe zaporedji so prikazane v tabeli 4.2. Število ločenih dvojic elementov, ki je skupno obema zaporedjema X in Y, je enako 6. Rezultat vrednotenja je torej razmerje 6/10 = 0,6.

Tabela 4.2: Ločene dvojice elementov za zaporedji X in Y.

Zaporedje	Ločene dvojice
X	(danes je), (danes resnično), (danes lep), (danes dan), (je resnično), (je lep), (je dan), (resnično lep), (resnično dan), (lep dan)
Y	(danes ni), (danes resnično), (danes lep), (danes dan), (ni resnično), (ni lep), (ni dan), (resnično lep), (resnično dan), (lep dan)

4.6 Metode razpletanja

Za razpletanje prepletenih sej smo razvili dva postopka. Pri razvoju prvega postopka smo se osredotočili na enostavnost in hitrost delovanja. Pri drugem postopku s preiskovanjem prostora stanj smo se osredotočili na čimboljši rezultat razpletanja.

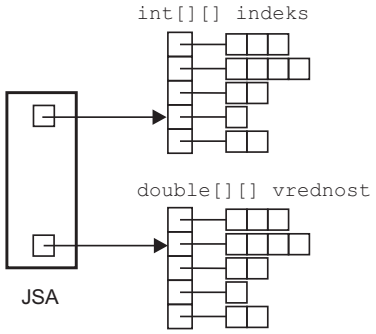
4.6.1 Uporaba markovskega modela

Postopek razpletanja temelji na preteklih akcijah spletnih uporabnikov. Obnašanje uporabnikov modeliramo s pomočjo markovskega modela, ki ga pred tem naučimo s podatki o čistih uporabniških sejah. Implementacija markovskega modela omogoča izdelavo markovskih modelov različnih redov. Markovski modeli različnih redov se med seboj razlikujejo po tem, koliko daleč v preteklost gledajo pri napovedovanju naslednje uporabniške akcije. Pri modelu prvega reda se naslednja stran uporabnika določi glede na zadnji obisk, pri modelu drugega reda na podlagi zadnjih dveh obiskanih strani, pri modelu k-tega reda pa na podlagi zadnjih k obiskanih strani. Sam koncept delovanja markovskih modelov različnih redov je podoben, zato bomo model predstavili na primeru markovskega modela prvega reda.

Markovski model razvit za podporo razpletanju v grobem vsebuje podatke o verjetnosti prehodov med stanji (spletnimi stranmi) in verjetnosti začetnih stanj (verjetnost, da je neka stran začetna stran). Verjetnosti prehodov med stanji se izračunajo

na podlagi podatkov o prehodu med stanji, ki se hranijo v matriki prehodov in na podlagi predznanja. Kot predznanje nam služi načrt spletnega mesta. Predstavljeno je v obliki usmerjenega grafa, ki hrani povezave med stranmi v spletišču. Matrika prehodov je odvisna od reda markovskega modela. Za model prvega reda z n možnimi izidi potrebujemo matriko velikosti $n \times n$, za model drugega reda $n \times n \times n$, za k-ti red pa $k+1$ dimenzionalno matriko. Prvih k dimenzij predstavlja zaporedje preteklih izidov (stanje), $k+1$ dimenzija pa število prehodov v naslednje stanje. Matriko prehodov lahko predstavimo z dvodimenzionalno matriko; k-dimenzionalno matriko z n možnimi izidi lahko predstavimo z dvodimenzionalno matriko velikosti $(\prod_{i=1}^{k-1} n) \times n$. Prva dimenzija vsebuje k-mestne kombinacije vseh izidov (stanje), druga pa število prehodov za vsak izid. Pri velikem številu možnih izidov n se z večanjem reda k markovskega modela, zaradi hitrega povečevanja števila stanj, zelo hitro povečuje matrika prehodov. Prostorska zahtevnost se povečuje tako hitro, da nam kmalu onemogoča dejansko implementacijo. V učnih podatkih se ne pojavijo vsa stanja, zato je veliko mest v matriki prehodov praznih (privzeta vrednost). Lahko se zgodi, da ima več kot 95% matrike prehodov prazne vrednosti, ki pa ravno tako zasedajo prostor v pomnilniku. Takšna matrika se imenuje redka matrika (ang. sparse matrix). Redka matrika ima običajno manj kot 10% neničelnih elementov.

Delo z matrikami je izjemno pomembno pri znanstvenih izračunih, zato je bilo razvitih več načinov za implementacijo redkih matrik tako, da se hranijo samo neničelne vrednosti [28, 58]. Matriko prehodov smo implementirali s pomočjo javanskih redkih tabel (ang. Java Sparse Array – JSA) [37]. Koncept JSA je možno implementirati s pojavitvijo programskih jezikov java in C#, ki imata večdimenzionalne tabele implementirane s pomočjo enodimenzionalnih tabel. Koncept je prikazan na sliki 4.13. Pri tem konceptu imamo dve tabeli, eno za shranjevanje referenc na tabelo vrednosti in drugo tabelo za shranjevanje referenc na ustrezne indekse tabel (po eden indeks za vsako vrstico).



Slika 4.13: Format predstavitve redke matrike v formatu JSA (Java Sparse Array).

Primer prikazuje matriko A in njeno predstavitev s pomočjo javanskih redkih matrik. Tabela `vrednost` v tabeli na drugem nivoju hrani neničelne elemente tabele A za vsako vrstico posebej. Tabela `indeks` pa hrani indekse na tiste stolpce, ki imajo

neničelno vrednost. Za prvo vrstico se na mestu z indeksom 0 nahaja element 10, na mestu z indeksom 4 pa element -2.

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix} \tag{4.23}$$

```
double[][] vrednost = {{10,-2},{3,9,3},{7,8,7},
                       {3,8,7,5},{8,9,9,13},{4,2,-1}};
int[][] indeks = {{0,4},{0,1,5},{1,2,3},{0,2,3,4},{1,3,4,5},{1,4,5}};
```

Pri učenju markovskega modela za razpletanje si pomagamo tudi s predznanjem o domeni. Predznanje je predstavljeno z načrtom strani (ang. site map). Načrt strani je sestavljen iz povezav med stranmi, ki so med sabo neposredno povezane s hiperpovezavami. Povezava med stranema S_i in S_j v načrtu strani pomeni večjo apriorno verjetnost prehoda med tema dvema stranema, kot če bi ti dve strani ne bi bili povezani med sabo. Glede na povezave med spletnimi stranmi izračunamo začetno verjetnost prehoda med stranmi $p_{ij}^{(0)}$, kjer i in j označujeta začetno in ciljno stran. Začetno verjetnost prehoda $p_{ij}^{(0)}$ izračunamo po formuli:

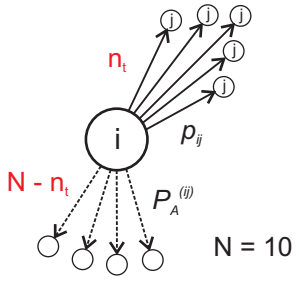
$$p_{ij}^{(0)} = \frac{1 - P_A^{(ij)}(N - n_t)}{n_t}, n_t \geq 1 \tag{4.24}$$

kjer j označuje vse strani, ki so povezane s stranjo i, N označuje število vseh strani, n_t pa število vseh izhodnih povezav iz strani i in $P_A^{(ij)} = 1/N^2$ enotno verjetnost prehoda med poljubnima dvema stranema (slika (4.14)). Pri tem predpostavljamo, da vedno obstaja zrcalen prehod iz s_i v s_i kar pomeni, da je n_t vedno večji od 0. Če med stranema i in j ni nobene povezave, se priredi verjetnost prehoda $P_A^{(ij)}$. Parameter $P_A^{(ij)}$ označuje apriorno verjetnost prehoda med poljubnima dvema stranema v načrtu spletne strani.

Naj bo vsaka seja predstavljena kot zaporedje spletnih strani $S = \{q_1, q_2, \ldots q_n\}$, kjer n označuje dolžino uporabniške seje. q_1 predstavlja začetno (vstopno) spletno stran in q_n zadnjo obiskano spletno stran uporabnika v tej seji. Pri izračunu verjetnosti prehoda med $q_{i-1} = s_j$ in $q_i = s_k$ lahko učne podatke kombiniramo z *m-oceno* (ang. m-estimate) [29]:

$$P(s_j \rightarrow s_k) = p_{jk} = \frac{(n_{jk} + m p_{jk}^{(0)})}{n_j + m} \tag{4.25}$$

kjer n_{jk} označuje število prehodov iz stanja j v stanje k, ki smo jih pridobili iz učnih podatkov. n_j je število obiskov strani j. m predstavlja utež, s katero določamo razmerje med apriornim znanjem (zajeto v načrtu spletne strani) in poznejšim



Slika 4.14: Shematski prikaz elemetov uporabljenih v enačbi (4.24).

znanjem, zajetim v učnih podatkih (uporabniške seje). Večji kot je m, večji je poudarek na apriornem znanju. Če imamo $m = 0$, potem popolnoma zanemarimo pomen apriornega znanja. V tem primeru se m-ocena spremeni v relativno frekvenco $p_{jk} = n_{jk}/n_j$.

4.7 Razpletanje z uporabo markovskega modela

Pri tej metodi smo uporabili markovski model in premočrten postopek razpletanja. Sam postopek temelji na verjetnostih prehoda med posameznimi stranmi. Pri razpletanju smo uporabili naučen markovski model. Markovski model je lahko prvega ali višjih redov. Pri razvoju postopka smo preizkušali razpletanje s pomočjo markovskega modela prvega in drugega reda. Kot je bilo navedeno pri opisu markovskih modelov, je pri modelih višjih redov potrebno imeti veliko množico podatkov, sicer imamo problem s pokritjem. Prepletena seja je sestavljena iz zaporedja strani. Nemogoče je na enostaven način ugotoviti, iz koliko elementarnih uporabniških sej je sestavljena. Ta podatek bomo izluščili med postopkom razpletanja. Vhod postopka je prepletena seja, ki je predstavljena kot zaporedje simbolov. Izhod postopka je seznam razpletenih prepletenih sej.

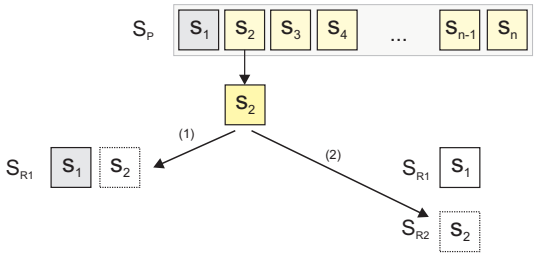
Postopek razpletanja temelji na dejstvu, da je prehod med dvema stranema $s_i \rightarrow s_{i+1}$ bolj verjetno pripada eni izmed sestavnih sej. Pri tem moramo paziti tudi na možnost, da je stran s_i začetna stran nove sestavne seje. Okviren postopek razpletanja je predstavljen v nadaljevanju. Postopek bomo prikazali pri uporabi markovsekga modela prvega reda. Predpostavimo, da imamo prepleteno sejo $S_p = [s_1, s_2, \ldots, s_n]$, dolžine n, ki je sestavljena iz k prepletenih sej dolžine n_1, n_2, $\ldots n_k$. V tem primeru velja $n_1 + n_2 + \ldots n_k = n$.

Sejo razpletemo tako, da se zaporedno pomikamo po vseh straneh s_i in strani dodeljujemo sestavnim sejam glede na verjetnosti prehodov. Postopek začnemo pri prvi strani s_1. Prva stran s_1 prepletene seje S_p je brez dvoma začetek ene izmed elementarnih uporabniških sej. Če je prepletena seja sestavljena samo iz ene seje, bo to edina začetna stran, če pa je sestavljena iz več sej, bomo v postopku razpletanja naleteli še na druge začetne strani. Razpletanje se torej začne pri strani s_2 –

razporeditev prve strani je trivialna. Pri naslednji strani (s_2) imamo dve možnosti: (i) stran lahko priredimo že obstoječi seji, ki vsebuje stran s_1, (ii) stran lahko predstavlja prvo stran nove razpletene seje. Za eno ali drugo možnost se odločimo na podlagi verjetnosti. Za stran s_2 preverimo, kakšna je verjetnost $P_{ZS}(s_2)$, da je s_2 začetna stran nove seje. Verjetnost začetnega stanja za določeno stran pridobimo iz naučenega markovskega modela. Iz markovskega modela pridobimo tudi podatek o verjetnosti prehoda med stranema $s_1 \rightarrow s_2$. Če velja neenakost

$$P_{ZS}(s_2)\,\omega > P(s_1 \rightarrow s_2), \tag{4.26}$$

potem ustvarimo novo razpleteno sejo, katere prva stran je s_2. V nasprotnem primeru se v tem trenutku edini razpleteni seji doda stran s_2. Rezultat razpleta po obdelanih dveh straneh prikazuje slika 4.15.



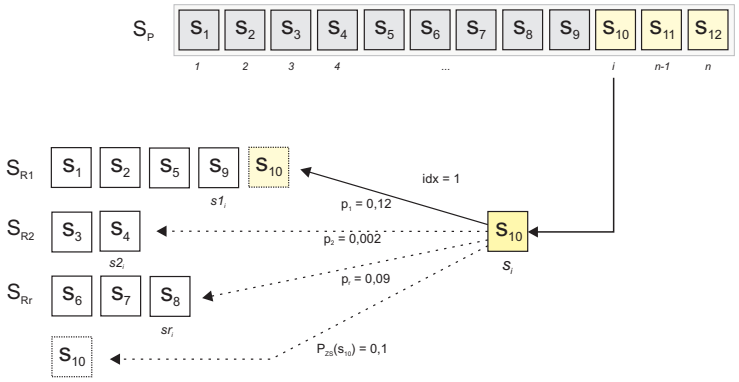
Slika 4.15: Posnetek stanja razpletanja. Pri razporeditvi strani s_2 imamo dve možnosti.

Verjetnosti začetka nove seje na neki strani s_x je pogosto prevelika v primerjavi z verjetnostjo nadaljevanja ene izmed že začetih sej. Kot rezultat razpletanja bi torej lahko dobili preveliko število razpletenih sej. Da se temu izognemo, je potrebno neenačbo (4.26) pravilno uravnotežiti. To storimo s faktorjem ω, za katerega velja $0 < \omega < 1$. Velikost faktorja ω je odvisna tudi od vira podatkov, zato ima faktor drugačne vrednosti za različne vire podatkov (npr. spletna trgovina, spletni IS). Faktor ω je potrebno predhodno ustrezno določiti na validacijski množici podatkov.

Postopek nadaljujemo za vse naslednje strani (s_i) prepletene seje S_p, dokler ne pridemo do stanja, ko druga razpletena seja pridobi prvi element (vstopno stran). Od tega trenutka naprej moramo za vse obstoječe začete razpletene seje preveriti, h kateri izmed njih stran s_i najverjetneje pripada. Seveda ne smemo pozabiti tudi na možnost, da je stran s_i začetna stran. Povedano formalno, recimo, da imamo v nekem trenutku r začetih razpletov. Zadnjo stran vsake izmed začetih razpletenih sej označimo z sj_i, kjer je $1 \leq j \leq r$. Za vsako stran s_i pridobimo iz markovskega modela verjetnost prehoda med zadnjimi stranmi razpletenih sej in trenutno stranjo v obdelavi s_i. Pridobiti moramo torej indeks seje idx z maksimalno verjetnostjo prehoda med sj_i in s_i:

$$idx = \arg\max_{j=1}^{r} P(sj_i \rightarrow s_i) \tag{4.27}$$

Najverjetnejši prehod med stranema je torej $s^i_{idx} \rightarrow s_i$. Preden stran s_i dodamo seji z indeksom idx, moramo preveriti še pogoj $P_{ZS}(s_i)\,\omega > P(s^i_{idx} \rightarrow s_i)$. Če pogoj ni izpolnjen, dodamo stran s_i seji z indeksom idx, sicer pa tvori s_i prvo, vstopno stran nove razplete seje z indeksom $r+1$. Na sliki 4.16 sta prikazani obe možnosti.



Slika 4.16: Postopek razpletanja prikazan na primeru.

Podobno postopek nadaljujemo za vse preostale strani s_i, dokler ne razporedimo zadnje strani (s_n), prepletene seje S_p. Predpostavimo, da smo v postopku razpletanja dobili m razpletenih sej. Če je postopek pravilno razpletel sejo, je $m = k$. Za vsako izmed m razpletenih sej $S_{r(i)}$ izračunamo verjetnost zaporedja njenih strani $P(S_{r(i)})$ po formuli:

$$P(S_{r(i)}) = P_{ZS}(s_1) \prod_{i=2}^{l} P(s_i) \tag{4.28}$$

kjer je l dolžina seje, $P_{ZS}(s_1)$ pa verjetnost, da se seja začne s stranjo s_1. Izračunamo še verjetnost celotnega razpleta tako, da pomnožimo med sabo verjetnosti vseh razpletenih sej:

$$P_{razplet} = \prod_{i=1}^{m} P(S_{r(i)}) \tag{4.29}$$

Postopek razpletanja je končan, razpletene seje se zapišejo na izhod.

Pri razpletanju uporabljamo naučen markovski model. V teoriji je lahko uporabljen markovski model k-tega reda. V praksi smo uporabili markovske modele prvega in drugega reda, praviloma smo uporabljali samo markovski model prvega reda. V primeru uporabe markovskih modelov višjih redov je potrebno pri razpletanju paziti tudi na izbor pravega reda modela, glede na trenutno stanje razpleta. Kot smo pri opisu markovskih modelov višjih redov videli, pri markovskem model k-tega reda sklepamo na naslednje stanje na podlagi k predhodnih stanj. Če nimamo znanih k predhodnih stanj, je potrebno uporabiti model ustrezno nižjega reda. Pri dodeljevanju naslednje nerazporejene strani (s_i) iz prepletene seje S_p eni izmed začetih razpletenih sej, primerjamo verjetnost prehoda med koncem seje in stranjo s_i. V primeru uporabe markovskega modela prvega reda, na trenutno stran s_i

sklepamo na podlagi zadnje strani razpletene seje, v primeru modela drugega reda na podlagi zadnjih dveh strani, itn. Če je dolžina razpletene seje manj kot k, potem je potrebno uporabiti markovski model nižjega reda tako, da je k manjši ali enak dolžini razpletene seje. Ni torej dovolj, če imamo naučen samo markovski model k-tega reda ampak moramo imeti naučeno celo verigo markovskih modelov od prvega do k-tega reda. Model nižjega reda moramo uporabiti tudi v primeru, ko so verjetnosti prehoda med začetimi razpletenimi sejami in s_i enake. Ta primer se lahko zgodi, če za zaporedja zadnjih k strani v začetih razpletenih sejah nimamo pokritja v markovskem modelu reda k in model vrne privzeto vrednost verjetnosti prehoda $P0$.

Razpletanje sej s pomočjo markovskega modela nam omogoča hitro in enostavno razpletanje prepletenih sej. Sam postopek razpletanja potrebuje linearno časovno zahtevnost $O(n)$, kjer je n dolžina prepletene seje. Pri tem predpostavljamo, da je število razpletenih sej majhno in precej manjše od dolžine prepletene seje. Tipično imamo od ene do tri prepletene seje. Za vsako stran prepletene seje s_i moramo namreč preveriti verjetnost prehoda med zadnjo stranjo začete razpletene seje in stranjo s_i. Časovna zahtevnost je torej $O(n \cdot r)$, kjer je n dolžina prepleta in r povprečno število razpletov. Vendar ker je r mnogo manjši od n, lahko r zanemarimo.

Razpletanje po tej metodi ne najde nujno prave rešitve (pravilnega razpleta). Razlog je v tem, ker metoda ne išče razpletenih sej z največjo verjetnostjo zaporedja. Povedano drugače, ne išče sej, ki so najbolj verjetne. Metoda se osredotoča na lokalno najverjetnejše povezave med stranmi. Lokalno zelo verjetni prehodi med stranmi pa ne vodijo nujno do seje kot celote, ki je najbolj verjetna.

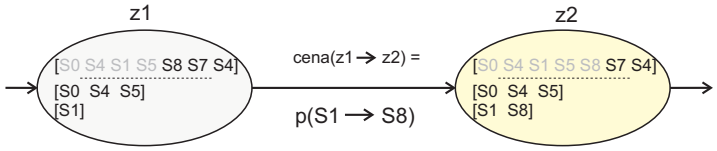
4.8 Razpletanje s preiskovanjem prostora stanj

Do problema razpletanja sej smo pristopili tudi s pomočjo preiskovanja prostora stanj. Pri tem smo morali problem razpletanja sej prevesti na problem iskanja v prostoru stanj. Prostor stanj je graf, katerega vozlišča ustrezajo problemskim situacijam, danemu problemu pa ustreza iskanje poti v grafu. Naloga razpletanja seje je ločiti zaporedje strani, ki tvori uporabniško sejo, v eno ali več krajših zaporedij strani. Postopek dodeljevanja strani poteka zaporedno od prve do zadnje strani tako, kot je uporabnik tudi izvajal zahteve na spletnem mestu. Strani v razpletenih sejah se morajo pojavljati v istem vrstnem redu kot v originalnem prepletu. Začnemo z izvorno prepleteno sejo in praznim seznamom razpletenih sej, končamo pa s prazno prepleteno sejo in seznamom razpletenih sej. V vsakem koraku izvedemo dodelitev naslednje strani prepletene seje eni izmed začetih razpletenih sej. Poiskati moramo zaporedje dodelitev strani prepletene seje razpletenim sejam. Nalogo razpletanja lahko preoblikujemo v problem preiskovanja alternativ v prostoru stanj. Stanje (problemska situacija) je predstavljena s prepleteno sejo $S_P = [s_1, s_2, \ldots, s_i, \ldots, s_n]$

dolžine n in r trenutno začetimi razpleti S_R.

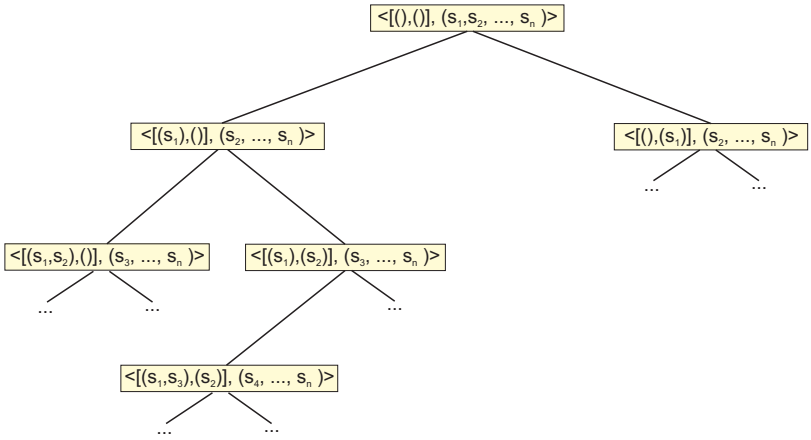
$$\begin{aligned} \text{stanje } Z &=< [S_{R(1)}, S_{R(2)}, \ldots, S_{R(r)}],\ S_P > \\ &=< [(sr_{1_1}, sr_{1_2}, \ldots, sr_{1_a}), \ldots, (sr_{r_1}, sr_{r_2}, \ldots, sr_{r_b})],\ (s_i, s_{i+1}, \ldots, s_n) > \end{aligned}$$

Začetno stanje je predstavljeno kot $ZS =< [(),(),\ldots,()], (s_1, s_2, \ldots, s_n) >$. Iz enega stanja v drugo prehajamo z veljavnimi prehodi ali akcijami. Slika 4.17 prikazuje primer dveh konkretnih stanj in akcijo, ki pretvori prvo stanje v drugo. Akcija pri



Slika 4.17: Primer prehoda med dvemi stanji za problem razpletanja sej.

razpletanju sej pomeni prestavitev naslednje, še ne razporejene strani s_i prepletene seje na konec ene izmed začetih razpletenih sej S_R. Sem spada tudi možnost začetka nove razpletene seje s številko $r+1$. Primer drevesa s prvimi nekaj stanji prikazuje slika 4.18. Končno stanje dobimo, ko dodelimo zadnjo stran s_n iz prepletene seje S_P eni izmed začetih razpletenih sej. Končno stanje lahko torej definiramo kot $KS =< [(sr_{1_1}, sr_{1_2}, \ldots, sr_{1_a}), \ldots (sr_{r_1}, sr_{r_2}, \ldots, sr_{r_b})], () >$. V listih drevesa dobimo vse particije množice strani prepletene seje. Listi drevesa predstavljajo končna stanja.



Slika 4.18: Graf prostora stanj za problem razpletanja ob predpostavki, da imamo samo dve razpleteni seji.

Med stanji (z akcijami) prehajamo za določeno ceno. Vsaka akcija ima ceno $c(z_1, z_2)$. V našem primeru je cena verjetnost prehoda med koncem ene izmed začetih prepletenih sej S_R in novo dodeljeno stranjo s_i. Če stran s_i predstavlja začetek

nove prepletene seje, potem je cena akcije enaka $c(z_1, z_2) = P_{ZS}(s_i)$, sicer pa je cena enaka verjetnosti prehoda med zadnjo stranjo začete prepletene seje in stranjo s_i. Iz tega sledi, da je verjetnost vsake izmed začetih razpletenih sej S_R enaka:

$$P(S_R) = P_{ZS}(sr_1) \prod_{i=1}^{a-1} P(sr_i \rightarrow sr_{i+1}) \tag{4.30}$$

kjer je P_{ZS} verjetnost začetnega stanja prve strani, a pa dolžina začetega razpleta. Določimo funkcijo $f(Z)$ kot kriterij (cenilko) za ocenitev stanja Z. Ceno določenega vozlišča Z dobimo tako, da pomnožimo med sabo vse cene poti od začetnega do trenutnega vozlišča Z. Funkcija $f(Z)$ je torej enaka produktu verjetnosti posameznih začetih razpletenih sej.

$$f(Z) = P(S_{R(1)}, S_{R(1)}, \ldots, S_{R(r)}) = \prod_{i=1}^{r} P(S_{R(i)}) \tag{4.31}$$

Cilj razpletanja je na podlagi prepleta S_P in naučenega markovskega modela najti končno vozlišče Z_K z največjo vrednostjo funkcije $f(Z_K)$. Poiskati moramo torej "najcenejšo" pot med začetnim in končnim vozliščem, ki predstavlja zaporedje premikov strani prepletene seje v razpletene seje. Cena rešitve je vsota cen povezav vzdolž rešitvene poti.

Z akcijami med sabo povezana problemska stanja tvorijo usmerjen graf, ki predstavlja prostor stanj. Vsako stanje je z drugimi stanji povezano na podlagi akcij delnega razpletanja. Predpostavimo, da imamo stanje Z. Stanje Z pretvorimo v nova problemska stanja Z_{nasl} tako, da nad stanjem Z izvedemo akcije dodeljevanja strani s_i na konec ene izmed začetih razpletenih sej S_R. Če ima stanje Z n_r začetih razpletenih sej, lahko iz Z kreiramo $n_r + 1$ naslednikov (Z_{nasl}). Prvo še nerazporejeno stran (s_i) v Z lahko dodamo na konec vsake izmed začetih razpletenih sej ali pa s_i tvori začetek nove seje. Cena povezave $c(z_1, z_2)$ predstavlja verjetnost prehoda med zadnjo stranjo seje S_R in stranjo s_i. Iz vsakega stanja Z imamo možnost prehoda v $n_r + 1$ naslednikov stanja. Iz vsakega stanja, ki pripadajo množici naslednikov Z_{nasl}, imamo spet možnost preiti v vsaj $n_r + 1$ stanj, itn. Število vozlišč grafa, ki predstavlja prostor stanj, se torej izjemno hitro povečuje z večanjem višine drevesa. Če je dolžina prepletene seje S_P enaka n, nam prostor stanj predstavlja drevo G z višino n. Za vsako dodeljeno stran s_i v prepleteni seji se višina drevesa poveča za ena. Vozlišča drevesa na istem nivoju imajo enak del razpletene seje. Število razvitih vozlišč drevesa po nivojih prikazuje tabela 4.3. Vidimo, da gre Bellova števila B_k, ki smo jih definirali v razdelku 3.4.3 in predstavljajo število particij množice velikosti k.

Pri reševanju problemov iskanja najkrajše (najcenejše) poti v usmerjenem grafu lahko velikokrat uporabimo algoritem za iskanje najkrajše poti v usmerjenem grafu G (t.j. Dijkstrov algoritem). V primeru razpletanja sej pa Dijkstrovega algoritma ne moremo uporabiti zaradi velike prostorske in časovne zahtevnosti. Velikost grafa, ki predstavlja prostor stanj, je enostavno prevelika. Pri razpletanju prepletene seje dol-

Tabela 4.3: Število razvitih vozlišč po nivojih za drevo, ki predstavlja prostor stanj.

Nivo	k	Št. vozlišč – B_k
1	0	1
2	1	1
3	2	2
4	3	5
5	4	15
6	5	52
7	6	203
8	7	877

žine 10 zahtev strani je prostor stanj sestavljen iz kar 142 417 možnih stanj. Namesto Dijkstrovega algoritma moramo torej uporabiti algoritem, ki zmanjša podgraf, ki ga algoritem mora pregledati za rešitev problema.

Če hočemo najti razplet z največjo verjetnostjo, moramo načeloma preiskati vse možnosti (celo drevo G). Vseh vozlišč drevesa je v skladu z vrednostmi v tabeli 4.3 enako $\sum_{k=0}^{n} B_k$. Preiskovanje vseh poti v drevesu G pa je izjemno zamudno zaradi razraščanja alternativ z vsakim novim nivojem drevesa, zato smo si pomagali s hevrističnim iskanjem. Namen hevrističnega iskanja je oceniti, katera vozlišča v množici kandidatov so bolj primerna za nadaljevanje in nadaljevati z iskanjem v smeri najbolj obetavnega vozlišča. Za iskanje poti v drevesu G od korena do enega izmed listov smo uporabili algoritem rekurzivnega iskanja po načelu najprej najboljši (RBFS). Algoritem smo izbrali zato, ker potrebuje samo linearno prostorsko zahtevnost, uspeli pa smo najti tudi učinkovito hevristično funkcijo za usmerjanje iskanja. Sam algoritem je podrobneje opisan v poglavju 4.3.5. Tukaj se bomo posvetili predvsem opisu hevristične funkcije in njeni implementaciji. Pri algoritmih za preiskovanje poti v prostoru stanj po definiciji seštevamo povezave med vozlišči in iščemo najcenejšo pot od začetnega do končnega vozlišča. V primeru razpletanja sej smo v enačbi (4.31) videli, da je cena poti od začetnega vozlišča do vozlišča Z produkt verjetnosti prehodov med stanji, mi pa iščemo razplet z največjo verjetnostjo. Zaradi lažje implementacije bomo problem prevedli na iskanje minimuma ter seštevanja cen povezav med vozlišči. Produkt posameznih korakov bomo logaritmirali in pomnožili z -1. Tako produkt verjetnosti postane vsota:

$$\begin{aligned} -\log(f(Z)) &= -\log(P_{ZS}(s_1)\prod_{i=1}^{n-1} P(s_i \to s_{i+1})) \\ &= -\log P_{ZS}(s_1) + \sum_{i=1}^{n-1} -\log P(s_i \to s_{i+1}) \end{aligned}$$

Od sedaj naprej bomo iskali minimum funkcije $-\log(f(Z))$ in seštevali cene korakov med vozlišči. Pri opisu usmerjenega iskanja smo videli, da ima vsako vozlišče

Z v prostoru stanj hevristično funkcijo f, ki ocenjuje ceno vozlišča Z. Ocena vozlišča upošteva predvidevanje, kakšna bo cena rešitve, če nadaljujemo iskanje do ciljnega vozlišča skozi vozlišče Z. Najbolj obetaven je kandidat iz množice možnih naslednjih vozlišč, ki minimizira vrednost funkcije f. Če gre pot skozi vozlišče Z, potem cenilko $f(Z)$ zapišemo kot vsoto dveh členov:

$$f(Z) = g(Z) + h(Z)$$

Funkcija $g(Z)$ je ocena najboljše poti od začetnega vozlišča do vozlišča Z, $h(Z)$ pa je ocena najboljše poti od Z do končnega stanja. Algoritem je pot od začetnega stanja do vozlišča Z že prehodil. Ta pot predstavlja najoptimalnejšo trenutno znano rešitev. Poti od Z do končnega vozlišča pa še nismo prehodili in ne vemo, kakšna je optimalna rešitev, zato moramo na nek način oceniti težavnost poti od Z do končnega vozlišča. Vrednost funkcije $g(Z)$ ni sporna. Pri razpletanju sej je $g(Z)$ enaka verjetnosti razpletenih sej v vozlišču Z. Verjetnost razpleta dobimo s množenjem verjetnosti prehodov med stranmi v razpletenih sejah. Ker smo razpletene seje zgradili postopoma po korakih s prehodi med stanji, je verjetnost razpleta enaka kar produktu prehodov med vozlišči v prostoru stanj, ki so pripeljale do vozlišča Z. Predpostavimo, da imamo v vozlišču Z stanje z R začetimi razpletenimi sejami. Vrednost funkcije $g(Z)$ je enaka:

$$g(Z) = -\log \sum_{r \in R} P_{ZS}(s_{r_1}) + \sum_{r \in R} \sum_{i=1}^{len(S_r)} -\log P(s_{r_i} \rightarrow s_{r_{i+1}})$$

Večjo težavo imamo z drugim členom $h(Z)$, saj vozlišč med Z in končnim vozliščem še ne poznamo. Ne vemo kako se bodo preostale strani v prepleteni seji (strani od s_i do s_n) porazporedile med začete razpletene seje. Ker nam ni znano zaporedje strani, ne moremo izračunati verjetnosti prehodov med njimi.

4.8.1 Hevristična funkcija

Kot smo navedli v razdelku 4.3.3, popolna hevristična funkcija daje optimistične ocene rešitve od trenutnega do prvega ciljnega vozlišča. Dejanska rešitvena pot mora biti torej daljša od tiste, ki nam jo vrne popolna hevristična funkcija. Ta pogoj smo upoštevali tudi pri izdelavi hevristične funkcije za problem razpletanja sej.

V prejšnjem razdelku smo videli, da je pri problemu razpletanja sej rešitvena pot od začetnega do končnega vozlišča enaka produktu verjetnosti prehodov med stanji. Na primer, če imamo začetno stanje s prepleteno sejo $[S1, S5, S0, S8, S4, S3]$, ki je sestavljena iz dveh elementarnih sej $[S1, S5, S4]$ in $[S0, S8, S3]$, potem je vrednost poti od začetnega vozlišča do končnega vozlišča z razpletom enaka:

$$P_{razpleta} = P_{ZS}(S1)P(S1 \rightarrow S5)P(S5 \rightarrow S4) \cdot P_{ZS}(S0)P(S0 \rightarrow S8)P(S8 \rightarrow S3)$$

Iščemo pot, ki nas pripelje do končnega vozlišča z največjim produktom verjetnosti prehodov med stranmi v razpletenih sejah. Hevristična funkcija $h(Z)$, ki zadošča

merilu popolnosti, mora torej razplet preostanka prepletene seje oceniti optimistično – maksimizirati mora produkt verjetnosti prehodov med še nerazporejenimi stranmi. Maksimizacija produktov verjetnosti je enakovredna minimizaciji vsote negativnih logaritmov teh verjetnosti.

Hevristik, ki zadoščajo pogoju popolnosti, imamo za določen problem lahko več. Med sabo se razlikujejo po uspešnosti usmerjanja iskanja do ciljnega vozlišča. V nadaljevanju si bomo pogledali nekaj hevristik za problem razpletanja sej. Zadnja izmed opisanih hevristik je najboljša, zato smo jo implementirali in uporabili pri procesu razpletanja.

Razvita hevristična funkcija je popolna v primeru, ko znamo z gotovostjo določiti začetna stanja prepletenih sej. Če začetnih stanj ni možno zanesljivo določiti, se namreč izkaže, da v postopku razpletanja dobimo preveč razpletenih sej.

Trivialna hevristika

Trivialna popolna hevristična funkcija za primer razpletanja sej je torej $h(z) = 1$. Trivialna hevristika predpostavlja, da je verjetnost prehoda med vsemi še nerazporejenimi stranmi enaka 1. Taka hevristična funkcija ne vpliva na vrednost cenilke $f(Z)$, saj je $f(Z)$ kar enaka $g(Z)$. Zadošča pogoju popolnosti, saj bo dejanski produkt verjetnosti prehodov med trenutno še nerazporejenimi stranmi v razpletenih sejah $h^*(Z)$ gotovo manjši od $h(Z)$. Verjetnosti prehodov med stranmi so tipično mnogo manjše od 1. Hevristična ocena $h(Z) = 1$ je enaka za vsa vozlišča, zato nima hevristične moči in ne nudi nobenega usmerjanja.

Največja verjetnost prehoda v stran

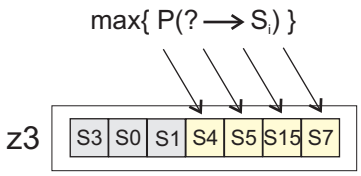
Hevristično funkcijo $h(Z)$ moramo čimbolj približati funkciji $h^*(Z)$, saj si s tem zagotovimo boljše usmerjanje. To lahko storimo tako, da pri izračunu hevristične funkcije za verjetnost prehoda v še nerazporejene strani namesto vrednosti 1 poiščemo bolj primerno vrednost. Večjo hevristično vrednost kot konstanta 1 (gotov prehod) ima največja verjetnost prehoda v stran s_i, $\max\{P(? \rightarrow s_i)\}$, ki je definirana kot

$$\max\{P(? \rightarrow s_i)\} = \max_{j=1}^{N_s}\{P(s_j \rightarrow s_i)\}, \tag{4.32}$$

kjer N_s predstavlja število vseh strani v sistemu. Hevristična ocena je torej produkt $\max\{P(? \rightarrow s_i)\}$ za vse še nerazpletene strani s_i. Izračun pokažimo še na primeru. Slika 4.19 prikazuje vozlišče z_3 z delnim razpletom, kjer so prve tri strani že razpletene, zadnje štiri strani pa še ne. Pri izračunu hevristične funkcije uporabimo produkt največjih verjetnosti prehoda v vsako izmed še ne razpletenih strani, torej:

$$h(z3) = \max\{P(? \rightarrow S4)\}\max\{P(? \rightarrow S5)\}\max\{P(? \rightarrow S15)\}\max\{P(? \rightarrow S7)\}$$

Taka hevristična funkcija je popolna, saj bo produkt verjetnosti razpletenih sej v končnem vozlišču kvečjemu enak ali manjši od hevristične ocene. Enak bi bil v primeru, ko bi dejanske vrednosti prehodov v razpletenih sejah sovpadale z največjimi



Slika 4.19: Vozlišče $z3$ z delno razpleteno sejo. Pri izračunu hevristične ocene upoštevamo strani iz prepletenega dela.

verjetnostmi prehodov v te strani. Recimo, da je razplet za prepleteno sejo na sliki 4.19 enak:

$$[S3, S0, S7] \text{ in}$$
$$[S1, S4, S5, S15]$$

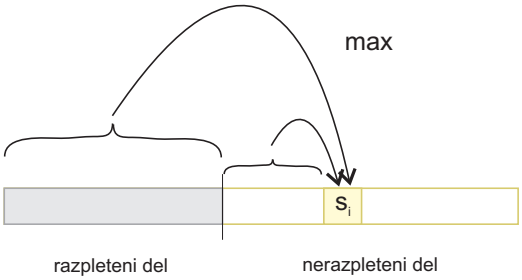
Hevristična ocena $h(z3)$ je enaka $h^*(z3)$, če velja:

$$P(S0 \to S7) = \max\{P(? \to S7)\} \text{ in}$$
$$P(S4 \to S5) = \max\{P(? \to S5)\} \text{ in}$$
$$P(S5 \to S15) = \max\{P(? \to S15)\}$$

Tipično pa ni tako, saj je vsaj ena ali več verjetnosti prehodov med stranmi $P(s_{i-1} \to s_i)$ znotraj ene izmed razpletenih sej manjših od največje verjetnosti prehoda v stran s_i. Hevristika je popolna, njena pomanjkljivost pa je, da za izračun max verjetnosti prehoda v stran s_i ($\max\{P(? \to s_i)\}$) upošteva vse strani v sistemu in ne samo tiste, ki se nahajajo v prepleteni seji. To pomeni, da je $\max\{P(? \to s_i)\}$ bližje vrednosti 1 (in posledično trivialni hevristiki), kot bi lahko bil. Največja pomanjkljivost hevristike je, da imajo vozlišča na istem nivoju drevesa enako hevristično oceno in zato ne nudi dovolj velike hevristične moči. To bomo izboljšali v naslednji hevristiki.

Upoštevanje možnih prehodov

Pomanjkljivost zgornje hevristike je ta, da pri izračunu največje verjetnosti prehoda v stran s_i upoštevamo prehode iz vseh strani v sistemu. Dovolj bi bilo, če bi pri izračunu $\max\{P(? \to s_i)\}$ upoštevali samo tiste strani, ki se nahajajo v prepleteni seji. Prehod v stran s_i je namreč možen samo iz ene izmed strani prepletene seje. Število strani, ki jih uporabimo za izračun $\max\{P(? \to s_i)\}$ pa lahko zmanjšamo še z upoštevanjem trenutnega stanja razpleta v vozlišču, za katerega računamo hevristično oceno. Vzemimo za primer stanje razpleta v vozlišču $z3$ na sliki 4.19. Stran $S5$ bomo v nadaljnji fazi razpletanja priključili eni izmed že razpletenih sej, ki že vsebujejo strani $S3, S0, S1$ ali pa eni izmed strani še ne razpletenega dela, ki se nahajajo pred stranjo $S5$. Pri izračunu $\max\{P(? \to S5)\}$ lahko torej upoštevamo samo



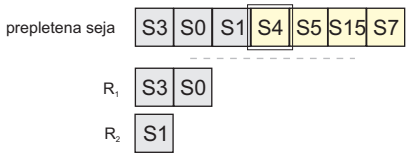
Slika 4.20: Način izračuna $\max\{P(? \rightarrow s_i)\}$. V stran s_i lahko pridemo samo iz ene izmed že razporejenih strani ali pa iz ene izmed predhodnih strani s_i še nerazporejenega dela.

verjetnosti prehodov iz strani $S3, S0, S1$ in $S4$. Shematično je postopek prikazan na sliki 4.20.

Hevristična funkcija je popolna, saj lahko uporabimo popolnoma enak razmislek kot pri zgoraj predstavljenih hevrističnih funkcijah. Razlika je ta, da smo funkcijo $h(Z)$ pomaknili bližje $h^*(Z)$. Takšna hevristična funkcija bolje usmerja, različna vozlišča imajo različne vrednosti hevristične funkcije. Izboljšamo jo lahko še tako, da upoštevamo dejansko zgradbo že razpletenega dela.

Končna hevristična funkcija

Zgornjo hevristično funkcijo lahko še malce izboljšamo. Osredotočimo se zopet na primer na sliki 4.19, ki prikazuje vozlišče $z3$. Razpleteni del seje je že formiran v eno ali več razpletenih sej, kot to vidimo na sliki 4.21. Očitno je, da lahko prvo

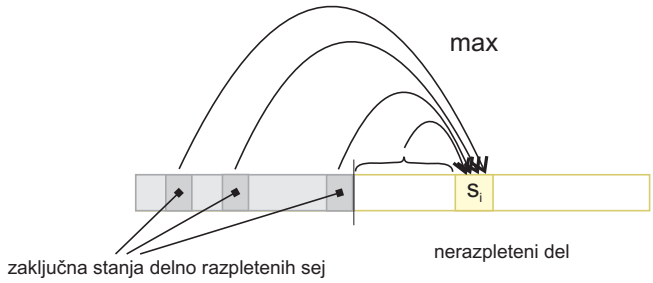


Slika 4.21: Delno razpletena seja.

stran $S4$ v še ne razpletenem delu dodamo samo na konec ene izmed razpletenih sej. Pri izračunu največje verjetnosti prehoda v stran $\max\{P(? \rightarrow s_i)\}$ nam torej ni potrebno upoštevati celotnega že razpletenega dela ampak samo končne strani že razpletenih sej. V konkretnem primeru za stran S_4 je to:

$$\max\{P(? \rightarrow S4)\} = \max\{P(S0 \rightarrow S4), P(S1 \rightarrow S4)\}$$

Podobno kot pri zgornji hevristiki moramo za vsako nerazporejeno stran s_i upoštevati tudi prehode iz predhodno ležečih strani v še nerazpletenem delu. Shematično je izračun največje verjetnosti prehoda v stran $\max\{P(\rightarrow s_i)\}$ prikazana na



Slika 4.22: Način izračuna $\max\{P(\rightarrow s_i)\}$ pri izračunu hevristične funkcije. Upoštevana je zgradba delno razpletenih sej.

sliki 4.22. Pogoj za popolnost je izpolnjen, saj za vsako še nerazporejeno stran s_i izberemo največjo verjetnost prehoda v to stran iz vseh realno možnih alternativ. Dejanska verjetnost prehoda med predhodno stranjo s_j in stranjo s_i v eni izmed razpletenih sej bo manjša ali enaka največji verjetnosti prehoda, uporabljeni pri izračunu hevristične ocene.

Predstavljeno hevristično funkcijo uporabljamo pri razpletanju sej, zato njen izračun formalneje opišimo. Naj s_i predstavlja prvo nerazporejeno stran v stanju Z, za katerega računamo hevristično oceno. Z s_m označimo stran, kjer je $i \leq m \leq n$, ki predstavlja trenutno stran v fazi simulacije razpleta za izračun hevristične ocene $h(Z)$, n pa označuje dolžino prepletene seje. Za vse trenutno še nerazporejene strani s_m moramo najti možne prehode v s_m in izbrati prehod z največjo verjetnostjo. Povedano drugače, najti moramo mesto v enem izmed razpletov, kamor stran s_m sodi z največjo verjetnostjo. Stran s_m lahko sledi stranem iz dveh skupin:

1. **Končne strani razpletov stanja** Z. Stran s_m se doda na konec ene izmed razpletenih sej (na primer seje d), ki jih trenutno imamo v stanju Z. Pri tem scenariju predpostavljamo, da se nobena stran s_l ($i \leq l < m$), ki se nahaja v prepletenem delu pred stranjo s_m, ne bo dodala na konec seje d (vse strani so se dodale na konec ene izmed ostalih razpletenih sej v množici S_R). Upoštevati moramo vse možnosti prehodov $S_{R_i} \rightarrow s_m$ in izbrati prehod z največjo verjetnostjo.

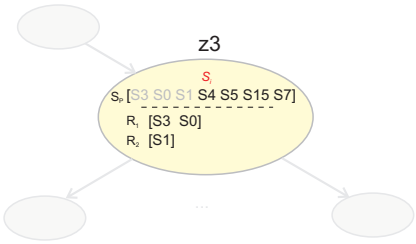
2. **Še nerazporejene strani stanja** Z. Stran s_m lahko sledi eni izmed strani, ki se v stanju Z še ne nahaja v začetih razpletih. s_m se torej doda na konec ene izmed razpletenih sej, ki pa se ji je pred tem že dodala ena ali več strani s_l z indeksom med i in m. Preveriti moramo torej verjetnost prehoda med vsemi možnostmi $s_l \rightarrow s_m$. Ker ne vemo, kako se bo dejansko izvedel razplet, moramo preveriti za vse predhodne strani s_l ($i \leq m$).

Označimo z $P_{max}(s_m)$ največjo verjetnost prehoda v stran s_m, ki jo dobimo na podlagi scenarijev opisanih v točki 1 in 2. Vrednost hevristične funkcije predstavlja produkt $P_{max}(s_m)$ za vse m med i in n.

Predpostavimo, da imamo primer vozlišča z R začetimi razpletenimi sejami. Označimo zadnje strani razpletov s_{r_z}. Prva nerazporejena stran prepletene seje je stran s_i. Vrednost hevristične funkcije h je enaka:

$$P_{max_r} = \max_{r \in R}(P(s_{r_z} \rightarrow s_m))$$
$$P_{max_s} = \max_{l=i}^{m-1}(P(s_l \rightarrow s_m))$$
$$h(Z) = \prod_{m=i}^{n} \max(P_{max_r}, P_{max_s}) = \prod_{m=i}^{n} P_{max}(s_m)$$

Izračuna hevristične funkcije h za določeno vozlišče prikažimo na primeru. Slika 4.23 prikazuje stanje z oznako $z3$, za katerega moramo izračunati hevristično oceno. $z3$ ima dva začeta razpleta, R_1 in R_2. Prve tri strani prepleta so že bile razporejene med razplete. Do ciljnega vozlišča je potrebno razporediti še strani $S4$, $S5$, $S15$ in $S7$. Vrednost hevristične funkcije je produkt prehodov z največjo verjetnostjo v vsako izmed teh strani. Postopek izračuna hevristične ocene je prikazan v



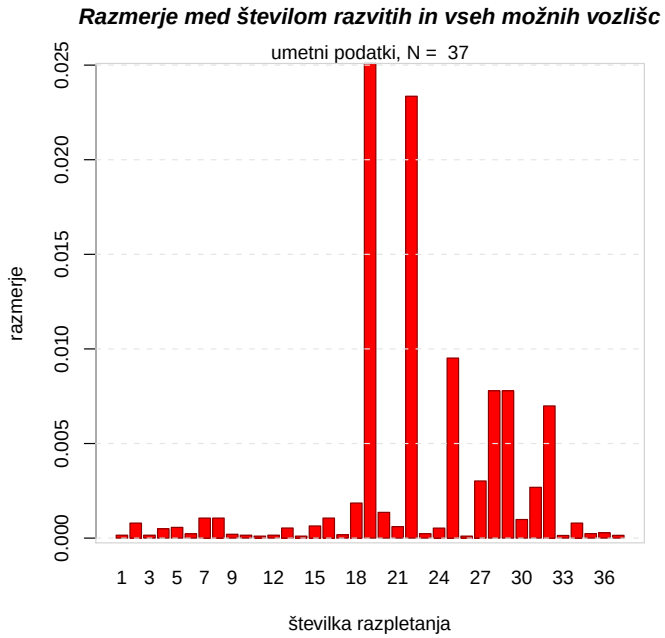
Slika 4.23: Stanje razpleta $z3$, za katerega izračunamo hevristično oceno.

tabeli 4.4. Še nerazporejene strani s_m so prikazane v prvem stolpcu tabele. Za vsako še nerazporejeno stran s_m se najprej izračuna verjetnost prehoda iz končnih strani $S0$ in $S1$ (2. stolpec tabele), nato pa še verjetnosti prehoda iz predhodno ležečih strani s_l v prepletu (3. stolpec tabele). Četrti stolpec tabele prikazuje največjo verjetnost prehoda iz točke 1 in 2 za vsak s_m. Vrednost h je produkt največjih verjetnosti prehodov (produkt po vrsticah).

Tabela 4.4: Izračun hevristične ocene za primer na sliki 4.23.

$\rightarrow s_m$	$(1 \rightarrow)$	$(s_l \rightarrow)$	$\max(P(1, s_l \rightarrow s_m))$	$h(z3)$
S4	S0 S1		0,2	0,2
S5	S0 S1	S4	0,4	0,08
S15	S0 S1	S4 S5	0,1	0,008
S7	S0 S1	S4 S5 S15	0,6	**0,0048**

Idealna hevristična funkcija najde rešitev linearno. To pomeni, da ne razvije nobenega vozlišča, ki ne bi bilo na rešitveni poti. V našem primeru bi idealna hevristična funkcija našla rešitev v n korakih, kolikor je višina drevesa, ki predstavlja prostor stanj. Višina drevesa je n, ker na vsakem nivoju razrešimo eno stran prepletene seje dolžine n. Hevristična funkcija h ni idealna in razvije večje število vozlišč, vendar učinkovito usmerja iskanje. To lahko vidimo na sliki 4.24, ki prikazuje rezultate razpletanja na umetnih podatkih in sicer razmerje med številom razvitih vozlišč in številom vseh vozlišč v prostoru stanj. Je popolna ter za tako definiran problem vedno najde optimalno rešitev. Prednost hevristične funkcije $h(Z)$ je tudi enostavnost in razumljivost.



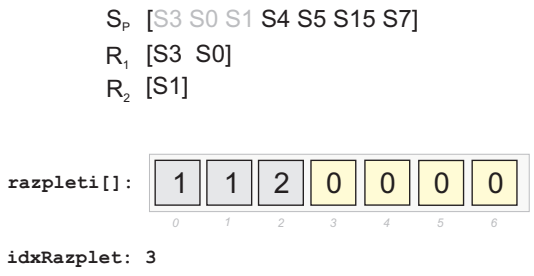
Slika 4.24: Razmerje med številom razvitih vozlišč in številom vseh vozlišč v prostoru stanj pri razpletanju na umetnih podatkih.

4.8.2 Implementacija RBFS

Za reševanje problema razpletanja sej s pomočjo usmerjenega iskanja, smo v prejšnjem razdelku določili ”pravili igre”. Definirali smo strukturo stanja za problem razpletanja sej, vpeljali začetno in končno stanje, določili ceno prehodov med stanji in vpeljali hevristično funkcijo. Za implementacijo konkretnega algoritma usmerjenega iskanja (v našem primeru RBFS) je treba implementirati razrede, ki določajo glavne elemente usmerjenega iskanja. Naša implementacija algoritma RBFS ima ogrodje, ki je splošno in je skupno vsem problemom iskanja. Deli, ki so specifični

za vsak konkreten problem, so določeni z vmesniki. Razredi `Stanje`, `GoalTest`, `SuccessorFunction`, `StepCostFunction` in `HeuristicFunction` implementirajo te vmesnike RBFS, in so specifični za problem razpletanja sej. Podrobneje jih bomo predstavili v nadaljevanju.

Razred `Stanje` implementira stanje razpletanja. Predstavlja trenuten razplet na poti od začetnega do končnega stanja. Algoritem RBFS lahko tekom iskanja najboljše rešitve generira veliko stanj. Stanje mora biti zato predstavljeno na način, da zaseda čimmanj prostora in omogoča enostavno generiranje naslednikov tega stanja. Razpletena seja je predstavlejna s tabelo celoštevilskega tipa `int[] razpleti` dolžine n, kjer je n dolžina prepletene seje. Števila v tabeli `razpleti` predstavljajo zaporedne številke prepletenih sej. Stran na mestu i v tabeli pripada seji z zaporedno številko $razpleti[i]$. Strani, ki še niso bile razvrščene v nobeno razpleteno sejo, imajo na svojih mestih vrednost 0. Pri določanju količine opravljenega razpletanja si pomagamo s spremenljivko `idxRazplet`, ki kaže na prvo stran, ki še ni bila dodeljena eni izmed razpletenih sej. Spremenljivka `idxRazplet` ni nujno potrebna, pomaga pa nam hitrejši dostop do prve razpletene seje. V javanskem objektu, ki predstavlja začetno stanje, so v tabeli `razpleti` vrednosti 0 na vseh mestih tabele. To pomeni, da nobena stran iz prepletene seje še ni bila dodeljena nobeni razpleteni seji. Slika 4.25 prikazuje vrednosti tabele `razpleti` za stanje $z3$ na sliki 4.23. Stanje vsebuje dve začeti razpleteni seji, ki ju označimo z indeksoma 1 in 2. Prve tri strani prepletene seje so bile že predhodno dodeljene razpletom. Prvi dve strani pripadata prvi seji, zato se na mestih 0 in 1 v tabeli nahaja številka 1. Tretja stran pripada drugi seji, zato je na tretjem mestu v tabeli `razpleti` vrednost 2. Indeks `idxRazplet` kaže na četro mesto v tabeli, kjer se nahaja prva še nerazporejena stran.



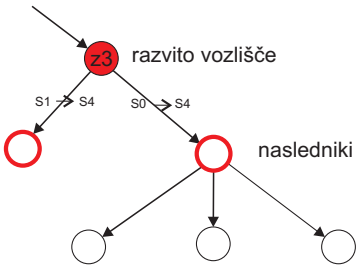
Slika 4.25: Vrednosti spremenljivk za stanje $z3$ na sliki 4.23.

Imena strani so shranjene v statični tabeli nizov z imenom `source`. Tabela imen strani se uporablja za to, da algoritem vrne sezname razpletenih sej s pravimi imeni strani. Pomembnen element razreda `Stanje` je metoda `vrniNaslednike()`. Metoda generira vsa možna naslednja stanja trenutnega stanja in jih vrne v formatu, ki jo zahteva ogrodje algoritma RBFS . Pri generiranju naslednikov trenutnega stanja nam pri testiranju lahko pride v pomoč omejitev števila možnih razpletenih sej. V ta namen se v razredu `Stanje` nahaja tudi statična spremenljivka `MAX_STEVILO_SEJ`,

ki se upošteva pri generiranju naslednikov stanja. Omejitev števila razpletenih sej smo s pridom uporabili pri testiranju hitrosti delovanja, številu razvitih vozlišč grafa in kakovosti razpletanja pri različnih vrednsotih tega parametra.

Razred `GoalTest` je zadolžen za implementacijo zaustavitvenega pogoja. Pri problemu razpletanja sej je ustavitveni pogoj enostaven. Ciljno vozlišče je doseženo, ko smo iz prepletene seje dodelili zadnjo stran eni izmed razpletenih sej. Razred `StepCostFunction` implementira funkcijo, ki vrača ceno povezave med dvemi stanji. Akcija je določena z dvemi stanji: stanje, ki predstavlja vir in ponor. Na podlagi obeh stanj ugotovimo parametre akcije in iz markovskega modela vrnemo ustrezno verjetnost. Razred `SuccessorFunction` vsebuje rutine za generiranje vseh naslednikov nekega stanja. Zaradi hitrosti, preprostosti in enostavnosti smo to funkcijo implementirali kot metodo znotraj razreda `Stanje`. Razred `SuccessorFunction` torej samo kliče metodo trenutnega objekta stanja in vrne rezultate v potrebnem formatu.

Razred `HeuristicFunction` je pomemben del RBFS za implementacijo hevristične ocene. Razred ima metodo `getHeuristicValue(Stanje Z)`, ki za stanje Z v parametru izračuna in vrne hevristično oceno, ki se uporabi pri ocenjevanju perspektivnosti poti skozi trenutno vozlišče. Sam postopek računanja hevristične ocene je predstavljen višje v razdelku 4.8.1. Slika 4.26 prikazuje mesto klica hevristične funkcije za vozlišče $z3$ v postopku iskanja. $z3$ je razvito vozlišče, za katerega so znani vsi njegovi nasledniki in akcije, da pridemo do njih.



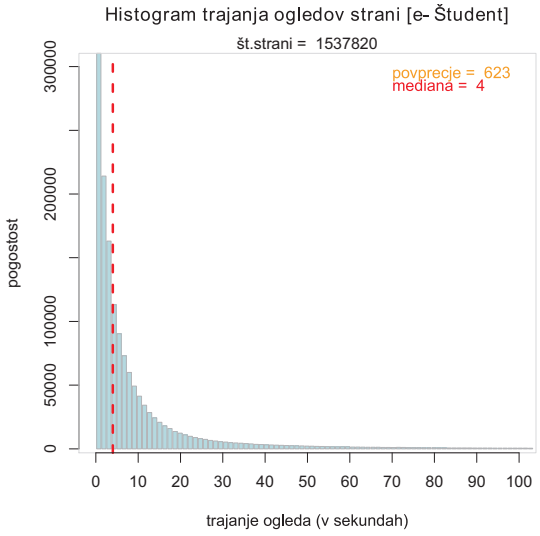
Slika 4.26: Mesto klica hevristične funkcije za razvito vozlišče $z3$.

4.9 Uporaba časovnih oznak pri razpletanju

Uporabnik pri deskanju po spletnem mestu išče določene podatke ali opravlja določeno nalogo. Pri tem izmenično proži zahteve za prenos spletnih strani in prebira vsebino. Uporabniška seja je torej sestavljena iz zaporedja strani, ki si sledijo v določenem časovnem zaporedju. Med dvema zaporednima stranema S_i in S_{i+1} mine določen čas t_{Si}, ki ga imenujemo *t*rajanje ogleda strani S_i. To je čas, ki preteče od uporabnikove zahteve za prenos strani S_i na odjemalca, do zahteve za naslednjo stran S_{i+1}. V tem času uporabnik ostaja na spletni strani S_i in pregleduje vsebino.

Trajanje ogleda strani je odvisno od vrste spletnega mesta (npr. spletna aplikacija ali novičarski portal) in vsebine spletnih strani (spletna stran z več teksta potrebuje več časa za branje).

Zanimalo nas je, kako je s trajanjem ogledov strani pri prepletenih sejah in ali je možno ta podatek uporabiti pri bolj natančnem razpletanju sej. Predvsem nas je zanimalo, če je trajanje ogleda strani t_{Si} odvisno od tega, ali naslednja stran S_{i+1} pripada isti seji ali ne. Podatke o trajanju ogleda strani smo pridobili s pomočjo spletnega podatkovnega skladišča za sistem e-Študent in spletno trgovino EnaA. Sliki 4.27 in 4.28 prikazujeta histograma trajanja ogleda strani za oba vira podatkov. Prikazana sta tudi povprečno trajanje ogleda strani in mediana.



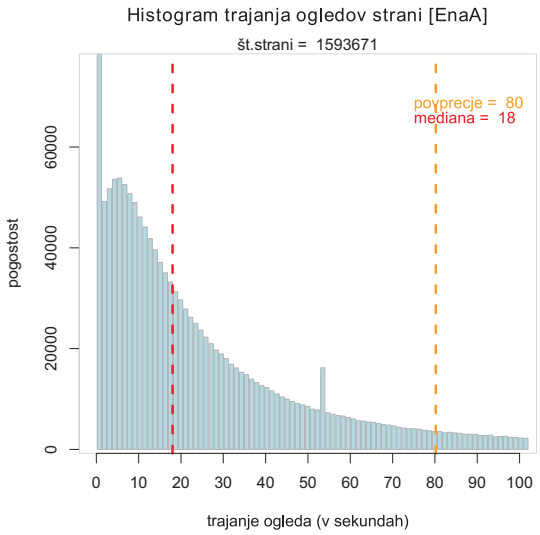
Slika 4.27: *Histogram trajanja ogledov strani za sistem e-Študent.*

Povprečna vrednost in mediana se za sistem e-Študent zelo razlikujeta. Razlog je v zelo različni uporabi sistema glede na uporabniško vlogo. Mediana trajanja ogleda strani za sistem e-Študent je 4 sekunde. Na grafu 4.27 vidimo, da je trajanje ogleda strani za veliko večino strani majhno, kar je pričakovano. Pri uporabi spletne aplikacije običajno uporabniki rutinsko izvajajo operacije, ogled strani je tipično zelo kratek. Povprečje trajanja ogleda strani dvignejo napredni uporabniki, ki so v sistemu prijavljeni cel delovni dan in se med končanimi posameznimi opravili ne odjavljajo iz sistema. Primera, ki dvigujeta povprečno vrednost trajanja ogleda strani, sta: čas mirovanja med dvemi opravili študentske pisarne, neaktivnost v času kosila, ipd. Naprednih uporabnikov je manj, zato ne vplivajo opazno na mediano, močno pa vplivajo na povprečni čas ogleda strani.

Pri spletni trgovini je med mediano in povprečnim trajanjem ogleda strani manjša razlika, kar pomeni, da imamo manj robnih primerov. Uporabniki se na straneh spletne trgovine zadržujejo dlje časa kot na straneh sistema e-Študent. Strani spletne trgovine vsebujejo več podatkov, zato uporabnik potrebuje več časa za branje,

drugačna je tudi narava uporabe obeh sistemov. Na sliki 4.28 vidimo, da se v okolici 50 sekunde pojavi manjši vrh, ki je verjetno posledica samodejnih periodičnih osvežitev strani.

Zaradi navedenega smo pri generiranju testnih podatkov za povprečen čas ogleda strani uporabili raje mediano kot povprečno trajanje ogleda strani.



Slika 4.28: *Histogram trajanja ogledov strani za spletno trgovino EnaA.*

Trajanje ogleda strani smo vključili v razpletanje tako, da smo verjetnost prehoda med dvema sosednjima stranema v prepleteni seji določali tudi na podlagi časa ogleda prve strani. Verjetnost prehoda med stranema S_i in S_{i+1} je torej enaka $P(S_i \rightarrow S_{i+1}, t_{Si})$, kjer t_{Si} predstavlja čas zadrževanja na strani S_i. Pri določanju funkcije t_{Si} smo uporabili naslednji razmislek.

Naj t_M označuje povprečen čas ogleda strani in t_{Si} čas ogleda strani S_i. Manjši kot je čas t_{Si}, manjša je verjetnost prehoda $P(S_i \rightarrow S_{i+1})$. Pri vrednosti $t_{Si} = 0$, je verjetnost prehoda $P(S_i \rightarrow S_{i+1}) = 0$. Bolj kot se t_{Si} približuje povprečnemu času ogleda strani t_M, večja je verjetnost prehoda. Ko prekoračimo čas t_M se verjetnost prehoda spet začne zmanjševati. Verjetnost prehoda naj se povečuje in zmanjšuje v skladu s funkcijo $f(t_{Si})$. Verjetnost prehoda med stranema S_i in S_{i+1} je določena na način:

$$P(S_i \rightarrow S_{i+1}, t_{Si}) = P(S_i \rightarrow S_{i+1}) f(t_{Si}) \tag{4.33}$$

$$f(t_{Si}) = \begin{cases} 0, & t_{Si} = 0, \\ P(t \leq t_{Si}), & t_{Si} \leq t_M, \\ 1 - P(t \leq t_{Si}), & t_{Si} > t_M. \end{cases}$$

kjer $P(t \leq t_{Si})$ predstavlja verjetnost, da je trajanje ogleda strani manjše ali enako

času t_{Si}. Verjetnost $P(t \leq t_{Si})$ je določena kot:

$$P(t \leq t_{Si}) = \int_{-\infty}^{t_{Si}} g(t)\,dt \tag{4.34}$$

kjer je $g(t)$ verjetnostna porazdelitvena gostota in pove, kolikšna je verjetnost trajanja ogleda neke strani točno določen čas t. Za $g(t)$ velja:

$$\begin{aligned} &\text{(a) } g(t) \geq 0 \;\; za \;\; \forall t, \\ &\text{(b) } \int_{-\infty}^{\infty} g(t)\,dt = 1. \end{aligned} \tag{4.35}$$

Teoretično bi funkcijo $f(t_{Si})$ izračunali na zgornji način, vendar v primeru razpletanja sej nimamo na voljo verjetnostne porazdelitvene gostote $g(t)$ za trajanje ogleda strani. Trajanje ogleda strani je izraženo v diskretnih časovnih intervalih s korakom ene sekunde (sliki 4.27 in 4.28). Za vsako trajanje ogleda t sekund vemo, koliko uporabnikov se je na strani povprečno zadrževalo točno t sekund. Od tod lahko izračunamo verjetnostno porazdelitev trajanja ogleda strani točno t sekund s korakom ene sekunde $P(t)$:

$$P(t) = \frac{n(t)}{\sum_{i=1}^{t_N} n(t)} \tag{4.36}$$

kjer t predstavlja čas v sekundah, $n(t)$ pogostost (število) ogledov strani točno t sekund in t_N zgornja meja trajanja ogleda strani v sistemu, ki jo še upoštevamo.

Integral (4.34) aproksimiramo z izračunom vsote verjetnosti trajanja ogleda strani $P(t)$ z intervalom ene sekunde.

$$P(t \leq t_{Si}) = \sum_{t=1}^{t_{Si}} P(t)\,dt \tag{4.37}$$

POGLAVJE 5

Opis testnih podatkov

5.1 Uvod

Pri razvoju metod za razpletanje sej smo uporabili podatke o klikotoku iz dveh virov: spletni študijski informacijski sistem e-Študent [61] in spletno trgovino EnaA [26]. Odločitev za ta dva vira podatkov je temeljila na dejstvu, da sta ta dva vira med seboj zelo različna. Sistem e-Študent je spletni informacijski sistem in ga uporabniki uporabljajo na drugačen način kot to počnejo pri spletni trgovini. Ker je klikotok odraz uporabnikovih akcij v brskalniku, so temu primerno različni tudi dnevniki spletnega strežnika obeh sistemov. Z razvojem in testiranjem na dveh tako različnih sistemih smo se skušali izognili težavi, da bi se razvita metoda preveč prilegala klikotoku določene vrste spletne strani. Postopki za razpletanje morajo z manjšimi odstopanji delovati na klikotoku iz poljubnega vira. V naslednjem delu si bomo pogledali lastnosti obeh sistemov in njihovih dnevniških podatkov.

5.2 Umetno generirani podatki

Podatki iz dnevniških datotek dejanskih strežnikov niso vedno najbolj primerni za razvoj in testiranje postopkov. Eden izmed razlogov je prevelika obsežnost podatkov. Težko je preveriti delovanje postopkov, če nimamo celovitega pregleda nad podatki bodisi zaradi količine bodisi zaradi obsežnosti problema. Poleg tega je potrebno tudi dobro poznavati podatke, nad katerimi izvajamo postopke. Zato smo postopke najprej preverili na umetnih podatkih.

Izdelali smo testno okolje za generiranje umetnih podatkov. Omogoča generiranje podatkov o klikotoku, ki so enostavnejši po obsegu in številu strani spletnega mesta, vendar so kljub temu podobni realnim podatkom. Na realnih podatkih smo

pridobili podatke o tipičnih uporabniških sejah in jih uporabili pri generiranju umetnih podatkov. Preverili smo število vseh strani spletnega mesta in kakšna je poprečna dolžina uporabniške seje. Ta dva parametra smo potem ustrezno zmanjšana uporabili pri generiranju umetnih podatkov. Tipično smo simulirali spletno mesto, kjer je bilo število vseh strani N_{strani} enako 20 oz. 30. Za generiranje spletnih sej in prehodov med posameznimi stranmi smo uporabili postopek, kjer smo najprej generirali:

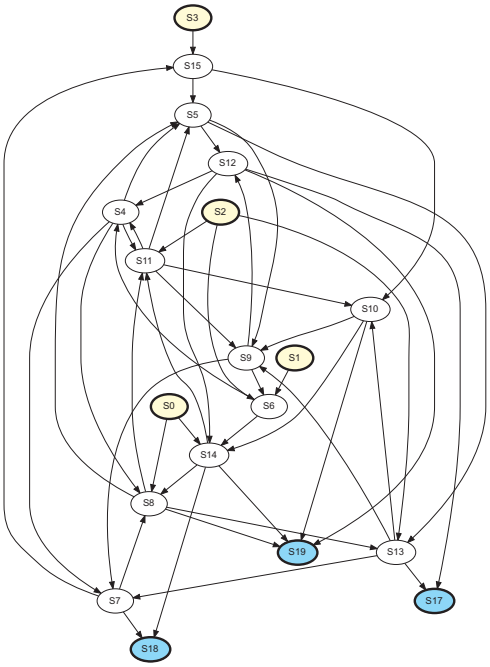
1. načrt spletnega mesta (spletišča),
2. tipične uporabniške poti (oz. tipske seje),
3. klikotok v (dnevniško) datoteko.

Načrt spletnega mesta je predstavljen z grafom in določa testno spletišče z vsemi stranmi in povezavami med njimi. Načrt spletnega mesta je določen s parametrom števila vseh strani N_{strani} in parametroma *minPovezav* ter *maxPovezav*, ki določata stopnjo vozlišča. Stopnja vozlišča pove, koliko ima lahko vsaka spletna stran najmanj oz. največ povezav s sosednjimi stranmi. Z upoštevanjem stopnje spletnega mesta dosežemo enakomerno razporeditev povezav po spletišču in se izognemo stanju, kjer bi nekatere strani imele preveč povezav druge pa nobene. Testne podatke smo generirali z vrednostmi parametrov $minPovezav = 2$ in $maxPovezav = 3$. Pri umetnih podatkih smo predpostavili, da uporabnik vedno vstopi na spletno stran pri eni izmed vnaprej določenih začetnih strani $S_{\text{začetna}}$ in sejo zaključi v eni izmed vnaprej določenih končnih strani $S_{\text{končna}}$. Število začetnih in končnih strani smo ustrezno prilagajali. Primer načrta strani za $N_{\text{strani}} = 20$ vidimo na sliki 5.1. Rumena vozlišča so štiri in predstavljajo začetne strani, modra tri vozlišča pa končne strani. Nobeno vozlišče ni z ostalimi povezano z manj kot eno in več kot štirimi povezavami.

Načrt strani je predstavljen v obliki usmerjenega grafa. Lahko pa ga preoblikujemo v format, ki ga uporablja orodje GraphViz in ga grafično predstavimo. Na podlagi načrta spletišča smo določili m tipičnih uporabniških poti, ki smo jih poimenovali *tipske seje*. Tipske seje predstavljajo načrt potovanja za uporabnike po spletni strani. V fazi generiranja klikotoka je pot vsakega uporabnika enaka eni izmed tipskih sej. Primera dveh tipskih sej na podlagi grafa na sliki 5.1 sta:

$$\text{tipska seja}_1\text{: S0} \rightarrow \text{S14} \rightarrow \text{S11} \rightarrow \text{S5} \rightarrow \text{S13} \rightarrow \text{S17}$$
$$\text{tipska seja}_2\text{: S3} \rightarrow \text{S15} \rightarrow \text{S5} \rightarrow \text{S12} \rightarrow \text{S4} \rightarrow \text{S7} \rightarrow \text{S18}$$

Na podlagi tipskih sej uporabnikov za dano spetišče smo zgenerirali klikotok v datoteko, ki predstavlja dnevniško datoteko spletnega strežnika. Za vsako tipsko sejo ts_i, kjer je $1 \leq i \leq m$, smo določili število uporabnikov n_i. To pomeni, da smo za vsako tipsko sejo ts_i v datoteko generirali n_i kopij. Format zapisa klikotoka je prikazan v tabeli 5.1.



Slika 5.1: Primer načrta spletišča z 20 stranmi, kjer so 4 strani začetne in 3 končne.

5.3 Sistem e-Študent

Sistem e-Študent je spletni študijski informacijski sistem namenjen opravljanju administrativnih nalog pri študijskem procesu. Na Univerzi v Ljubljani se na več fakultetah uporablja od leta 2003. V letu 2005 je sistem uporabljalo 16 fakultet. V tem delu smo uporabili dnevniške datoteke strežnika Fakultete za računalništvo in informatiko, ki ga uporabljajo tri fakultete: Fakulteta za računalništvo in informatiko, Fakulteta za elektrotehniko (FRI) in Fakulteta za strojništvo. Uporabnik se mora prijaviti v sistem, da lahko uporablja sistem. Uporabniki vstopajo v sistem preko skupne vstopne točke. To nam omogoča, da lahko precej bolj točno določimo začetek seje. Uporabnik naj bi se po koncu dela tudi odjavil iz sistema. V tem primeru imamo v dnevniški datoteki podatek tudi o odjavi iz sistema. Tipične uporabniške poti so torej dobro definirane. To dejstvo nam pomaga pri določanju začetka in konca seje pri postopku predprocesiranja in osejevanja podatkov. Seveda lahko uporabnik enostavno zapre okno brskalnika. V tem primeru moramo na konec seje uporbnika sklepati glede na časovno prekinitev. Slika 5.2 prikazuje oba načina zaključka seje. Tipično je časovna prekinitev postavljena na 20 do 30 minut. Za sistem e-Študent smo to časovno mejo dvignili na 24 ur. Za to smo se odločili, ker imajo nekateri uporabniki daljša obdobja neaktivnosti med zahtevami strani (npr. profesorjev sproten vnos ocen študentov na ustnem izpitu, narava dela zaposlenih v študijskem referatu).

Glavna težava pridobivanja podatkov o klikotoku iz dnevnika spletnega stre-

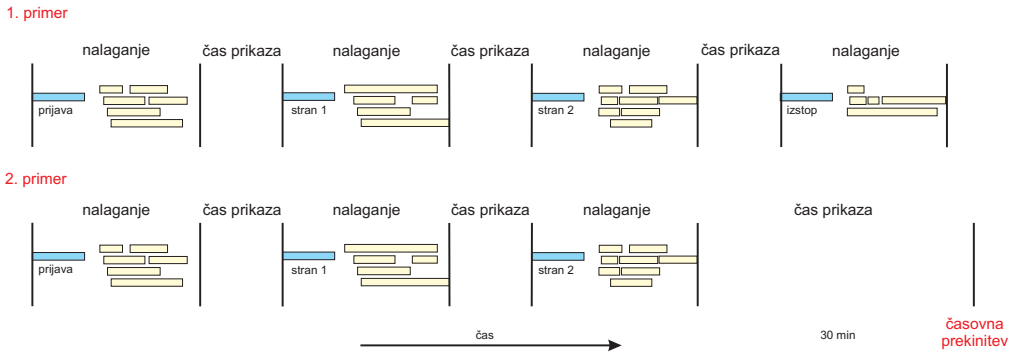
Tabela 5.1: Umetno generiran klikotok za dva uporabnika, ki se obnašata v skladu s tipskima sejama 1 in 2.

uporabnik	datum	čas	ime strani
[user61]	2007.06.22	06:29:49	S0
[user61]	2007.06.22	06:30:48	S14
[user61]	2007.06.22	06:31:00	S11
[user61]	2007.06.22	06:32:38	S5
[user61]	2007.06.22	06:33:54	S13
[user61]	2007.06.22	06:34:04	S17
[user71]	2007.06.22	02:51:43	S3
[user71]	2007.06.22	02:53:19	S15
[user71]	2007.06.22	02:54:59	S5
[user71]	2007.06.22	02:56:41	S12
[user71]	2007.06.22	02:57:47	S4
[user71]	2007.06.22	02:58:10	S7
[user71]	2007.06.22	02:58:53	S18

žnika je odsotnost podatka o uporabnikovi seji. Posamezne zahteve med seboj niso neposredno povezane. Pri združevanje zapisov v seje lahko pride do napak. Podatke o klikotoku bi lahko namesto iz dnevnika spletnega strežnika pridobili neposredno iz aplikacijske plasti. Aplikacijo e-Študent bi dopolnili z modulom za beleženje uporabnikovih akcij neposredno v podatkovno bazo (tabela uporabnikovih zahtev strani). Vsakemu uporabniku se ob prijavi samodejno določi enolična številka aplikacijske seje, ki bi jo lahko uporabili tudi za številko seje klikotoka. V dnevnik uporabnikovih zahtev bi lahko zapisovali samo za nas koristne podatke o zahtevah strani. Vsako zahtevo bi lahko natančneje opisali, npr. ali gre za osvežitev strani. Z beleženjem dogodkov na aplikacijski plasti bi se izognili potrebi po osejevanju in težavam z roboti spletnih iskalnikov (ang. web robot). Ravno tako ne bi prišlo do težav s prepletenimi sejami.

Beleženje uporabnikovih zahtev na aplikacijski plasti ima tudi slabosti. Celotno aplikacijo je potrebno ustrezno prilagoditi, kar ni vedno enostavno ali možno. Potreba po analiziranju obnašanja uporabnikov ponavadi nastane šele, ko aplikacija že nekaj časa nameščena in predana v uporabo, razvojna skupina pa ni več na voljo. Prilagoditve so torej težje izvedljive. Če uporabljamo kot osnovo (podlago, ogrodje) sisteme za hitrejšo gradnjo aplikacij (npr. Oracle Portal), ne moremo vedno posegati v vse dele sistema. Na aplikacijskem nivoju tudi ne moremo beležiti zahteve za strani, ki niso generirane dinamično. Primer so razne statične strani html, kot so: vstopne strani, datoteke pomoči, navodila za prijavo v sistem ipd.

Uporabnik se lahko prijavi v sistem v eni izmed štirih uporabniških vlog, kot študent, profesor, študentska pisarna ali vzdrževalec. Prepletene seje na sistemu nastanejo zato, ker se uporabnik v dveh oknih brskalnika lahko prijavi z različnimi



Slika 5.2: Prikaz dveh načinov zaključka uporabniške seje. V prvem primeru se seja konča z odjavo iz sistema. V drugem primeru se uporabniška seja zaključi s časovno prekinitvijo.

uporabniškimi vlogami in v obeh brskalnikih izvaja določene naloge. Tipičen tak primer je uporabnik, ki je v dveh oknih brskalnika prijavljen v vlogi študentske pisarne in vzdrževalca. Uporabnik v vlogi študentske pisarne izvaja neko nalogo, vendar je ne more uspešno izvesti ker manjka določen podatek v šifrantih podatkovne baze. V novem oknu se prijavi kot vzdrževalec in ustrezno ažurira šifrant, nato pa nadaljuje delo v oknu, kjer je prijavljen kot študentska pisarna. Na ta način nastanejo prepletene seje. Aplikacija uporablja za vsako shemo okrog 180 strani (cca. 170 funkcij sistema). V dnevniški datoteki spletnega strežnika FRI je torej za vse tri namestitve torej najdemo 550 različnih strani.

Dnevniški zapisi spletnega strežnika e-Študent uporabljajo osnovni format CLF (NCSA Common Log Format) [59]. Vsak zapis v dnevniški datoteki vsebuje podatke, ki jih vidimo v tabeli 5.2. Ogrodje aplikacije predstavlja Oracle Portal, ki nudi vrsto prednosti. Pri vsaki uporabniški zahtevi Oracle Portal povzroči zapis več različnih vrstic v dnevniško datoteko spletnega strežnika, kot bi se to zgodilo pri klasičnem spletnem strežniku. Identifikacija in odstranjevanje takšnih podvojenih zapisov predstavlja dodatno težavo pri procesu procesiranja podatkov iz dnevnika spletnega strežnika. V tabeli 5.2 predstavlja atribut *authuser* uporabniško ime uporabnika. To nam omogoča, da spletne seje lahko povežemo z dejanskimi uporabniki sistema.

Format CLF dnevnika spletnega strežnika zaradi zelo skopega števila zabeleženih atributov zahteve predstavlja precej velike omejitve pri postopku predprocesiranja podatkov. Manjkajoči in zelo uporaben je podatek o podpisu odjemalca (ang. web browser), ki je zahteval spletno stran (ang. user agent). Pri tem nas ne zanima toliko, kakšne spletne brskalnike uporabljajo uporabniki. Podatek nas zanima zaradi identifikacije in odstranjevanja zahtev robotov spletnih iskalnikov, ki se tudi nahajajo v dnevniški datoteki spletnega strežnika. Vsak robot iskalnika ima določen podpis, ki se pri razširjenem formatu ECLF shrani v atribut *user-agent*. Pri formatu CLF ta podatek manjka in spletnih robotov ni možno identificirati. Obstaja sicer

več hevristik, kako kljub odsotnosti tega podatka identificirati uporabnike – robote, vendar so lahko nezanesljive. Ena izmed takih hevristik uporablja prepoznavanje robotov po njihovem prečesavanju spletnega mesta. Roboti iskalnikov namreč v nekem časovnem obdobju generirajo zahteve za vse strani na spletnem mestu. V takem primeru je velika verjetnost, da gre za robota spletnega iskalnika.

Pri dnevniku spletnega strežnika e-Študent težava z roboti spletnih iskalnikov kljub temu ni tako velika. Razlog tiči v dejstvu, da se je za uporabo sistema e-Študent potrebno v sistem prijaviti. Spletni iskalniki pa nikoli ne uspejo čez postopek prijave. Veliko večino njihovih zahtev torej zavržemo, ko upoštevamo prag najmanj 4 strani za dolžino uporabniške seje.

Tabela 5.2: Format dnevniške datoteke sistema e-Študent.

Podatk. element	**Opis elementa**
host	naslov IP ali ime domene
ident	dodaten ID
authuser	uporabniško ime, ki se je uporabilo v zahtevi
time	čas zahteve na strežniku v formatu CLS
request	prva vrstica zahteve odjemalca
status	status koda, ki se vrne odjemalcu
bytes	količina podatkov, ki se vrne odjemalcu (v bajtih), brez glave HTTP

V postopku predprocesiranja podatkov smo dobili veliko sej dolžine manj kot štiri, ki pa za nas niso bile preveč zanimive. Za izvedbo neke naloge v sistemu je potrebno izvesti več klikov. Seje običajno pripadajo eni iz med množic: osirotele strani zaradi napake v postopku osejevanja, zahteve spletnih pajkov ali neuspešen postopek prijave v sistem. Seje dolžine ena običajno pomenijo, da ima uporabnik v brskalniku našo stran nastavljeno kot domačo stran. Takšne seje smo sicer shranili v podatkovno skladišče, nismo pa jih uporabili v postopkih razpletanja sej. Razlog je preprost, saj je seja dolžine štiri težko prepletena, samo postopek prijave v sistem terja sejo dolžine vsaj tri.

Ukvarjali smo se samo s sejami, ki so daljše ali enake 4 zahtevam in krajše ali enake 100 zahtevam v seji. Daljše ali krajše seje ne predstavljajo tipičnega obnašanja uporabnikov in smo jih zato opustili. V tabeli 5.3 vidimo povprečno dolžino uporabniških sej. Zgornja meja za dolžino seje je postavljena precej višje kot znaša povprečna dolžina seje, ker smo hoteli zajeti napredne uporabnike in vzdrževalce, ki kreirajo daljše uporabniške seje. Število uporabniških sej naprednih uporabnikov, ki so potencialni generatorji daljših in prepletenih sej, je precej manjše kot število sej navadnih uporabnikov, ki kreirajo krajše seje. To se seveda zrcali v precej nižji povprečni dolžini seje.

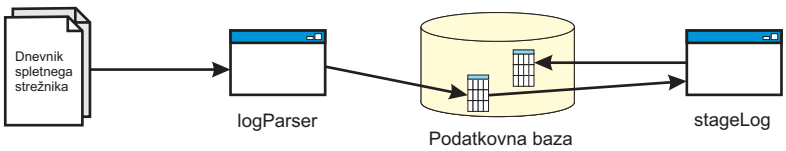
Kljub temu, da se spletni študijski IS e-Študent veliko uporablja, je število sej relativno majhno v primerjavi s spletno trgovino ali portalom z novicami. Razlog

je relativno majhno število uporabnikov in narava uporabe informacijskega sistema. Uporabniki spletni informacijski sistem uporabljajo, ko morajo izvesti določeno nalogo, spletno trgovino ali novičarski portal pa ponavadi prebrskajo pogosteje. Tabela 5.3 prikazuje podatke o sejah sistema e-Študent po letih; upoštevane so seje dolžine med 4 in 100 zahtev. Prvi stolpec predstavlja koledarsko leto, drugi stolpec prikazuje število vseh zahtev strani, tretji število sej, četrti pa povprečno število dolžine uporabniške seje. Iz podatkov je opaziti, da se uporaba sistema iz leta v leto rahlo povečuje. Veča se tako število zahtev za strani kot tudi število uporabniških sej. Povprečna dolžina seje z leti ostaja na enaki vrednosti, okrog 16 strani na sejo. Uporaba sistema se zelo poveča pred in v času izpitnih obdobij. Samo v prvih dvajsetih dneh januarja 2009 je bilo kreiranih več kot 16.000 uporabniških sej.

Tabela 5.3: Število zahtev strani, število sej in povprečna dolžina seje za IS e-Študent.

Leto	Število zahtev	Število sej	Povp. dolžina seje
2006	3.150.492	189.473	16,6
2007	3.619.890	220.193	16,5
2008	4.254.829	257.512	16,5

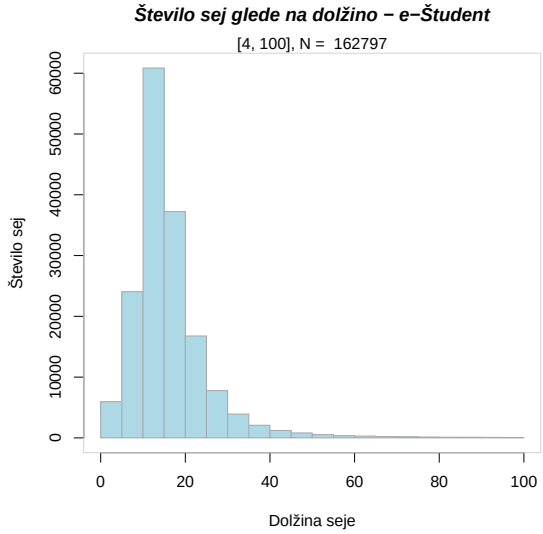
Podatki iz dnevnika spletnega strežnika e-Študent se prenesejo v področje za čiščenje in preoblikovanje podatkov. Tu jih potem lahko uporabimo za dodatne obdelave (npr. razpletanje prepletenih sej) ali pa jih prenesemo v podatkovno skladišče. Za zajem podatkov v dnevniški datoteki, čiščenje, preoblikovanje in osejevanje zapisov smo napisali lastne programe v jeziku C#. Postopek prenosa podatkov o klikotoku v podatkovno bazo področja za čiščenje in preoblikovanje podatkov je prikazan na sliki 5.3. Postopek smo razdelili na dva dela z namenom hitrejše in postopne obdelave podatkov. Prvi program, z imenom *logParser*, podatke zajame iz dnevniške datoteke, zavrže nepomembne zapise, izvede osejevanje in preveri integriteto sej. V skladu s parametri osejeni podatki se prenesejo v ustrezno tabelo v podatkovni bazi. Drugi program (*stageLog*)je zadolžen za poenotenje imen strani (ista stran v dnevniški datoteki ima lahko več imen), brisanje odvečnih zapisov in odpravljanje napak pri osejevanju.



Slika 5.3: Postopek za procesiranje podatkov o klikotoku za sistem e-Študent.

Slika 5.4 prikazuje razporeditev števila sej glede na dolžino seje za e-Študent. Glavnino vseh sej najdemo med dolžino 5 in 25 stranmi. Največje število sej je

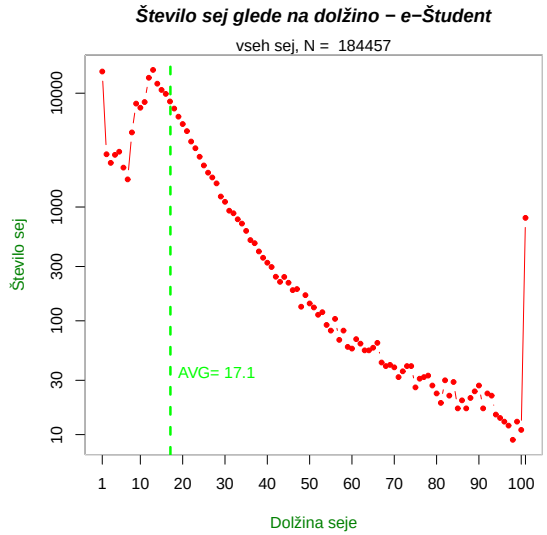
dolžine med 10 in 20 stranmi. Sorazmeroma veliko je tudi sej dolžine med 5 in 10 stranmi. Seje, ki so daljše od 30 strani, predstavljajo samo manjši delež. Število tudi zelo hitro pada z večanjem dolžine seje, tako da je dolžine več kot 50 samo nekaj sej. Slika 5.5 prikazuje število vseh sej ne glede na dolžino. Kot smo že omenili, je



Slika 5.4: Histogram števila sej glede na dolžino sej sistema e-Študent. Število vseh sej v vzorcu je 162.797.

veliko sej dolžine 1, kar se lepo vidi. Veliko sej je tudi daljših od 100 zahtev. Takšne seje so lahko posledica neustrzne rekonstrukcije nekaterih spletnih sej iz dnevnika spletnega strežnika. Nekaj sej med njimi gotovo pripada naprednim uporabnikom, katerih aktivnost na strežniku tvori zelo dolge seje.

Obdelani, prečiščeni in osejeni podatki se shranijo v podatkovni bazi v področju za čiščenje in preoblikovanje podatkov, od koder se potem vnesejo v podatkovno skladišče. Tabela 5.4 prikazuje primer uspešno rekonstruirane seje brez napak, ki pripada uporabniku v vlogi profesorja. Seja je neprepletena (čista), dolga 24 zahtev in se zaključi z odjavo iz sistema. Prvi stolpec predstavlja zaporedno številko zahteve strani. Drugi in tretji stolpec predstavljata datum in čas zahteve, v četrtem stolpcu se nahaja šifra uporabnika, v petem pa polno ime zahtevane strani. Zadnji stolpec tabele pove ali je bila stran klicana prvič (vrednost 0) ali pa je bila osvežena (vrednost 1). Do osvežitve strani pri spletnih aplikacijah lahko pride zaradi spreminjanja parametrov vnosnega obrazca in s tem potrebe po ponovnem generiranju strani na strežniku. Imena strani v rekonstruiranih sejah so popravljena tako, da so čimbolj opisna in berljiva. Iz seje v tabeli 5.4 lahko rekonstuiramo uporabnikovo nalogo. Uporabnik se je prijavil v sistem e-Študent, pregledal kandidate pisnega izpita. Nato je vnesel rezultate pisnega izpita in zgeneriral poročilo v formatu pdf. Nato se je odjavil iz sistema.



Slika 5.5: Števila vseh sej glede na dolžino za sistem e-Študent.

Bralec bi iz imen strani lahko napačno sklepal, da bi razpletanje strani lahko izvedli s pomočjo imen strani vendar temu ni tako. Različne vloge si med seboj delijo strani z istimi imeni, torej ne moremo sklepati, kateri vlogi (in s tem seji) neka stran pripada.

5.4 Spletna trgovina

Za drugi vir podatkov o klikotoku smo izbrali eno izmed najbolj priljubljenih spletnih trgovin – trgovino EnaA podjetja Gambit. Spletna trgovina EnaA je namenjena veliko širšemu krogu ljudi ima zato veliko večje število uporabnikov. Dnevno je zgeneriranih okrog 10000 sej. Uporabniki spletne trgovine so v veliki večini anonimni. Po spletnih straneh trgovine se lahko sprehajajo brez predhodne prijave v sistem z uporabniškim imenom in geslom. Tudi če pravilno rekonstruiramo uporabniške seje, ne moremo ugotoviti identitete uporabnikov, ki so naredili pot po spletnem mestu. Prijava uporabnika je nujna šele, ko se uporabnik odloči za nakup. V sistemu e-Študent smo za veliko večino sej lahko ugotovili tudi identiteto uporabnika. Ker prijava v sistem ni potrebna, lahko zahteve prihajajo od kjerkoli.

Odsotnost zahteve za prijavo v sistem pomeni tudi odsotnost skupne vstopne točke uporabnika. Vstopna stran je lahko katerakoli spletna stran trgovine. Na eni strani imamo lahko uporabnika, katerega prva zahtevana stran je globoko v hierarhiji spletnega mesta. Bodisi ima uporabnik to spletno stran v zaznamkih bodisi je na našo spletno stran prišel preko rezultatov spletnega iskalnika. Takšen obisk je lahko tudi posledica vključenih elementov (npr. pasic) na nekem drugem spletnem mestu (ang. deep link). Drugo skrajnost predstavlja uporabnik, ki pride na spletno

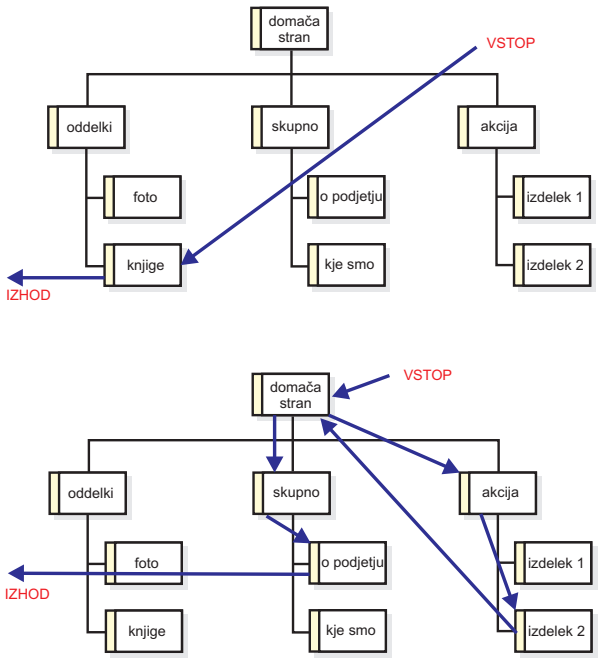
Tabela 5.4: Primer čiste seje sistema e-Študent v podatkovni bazi. Seje pripada uporabniku v vlogi profesor.

#	datum	čas	upor.	ime strani	osv.
1	29.8.2007	16:35.57	000486	/index.html	0
2	29.8.2007	16:35.59	000486	PORTAL30.wwsec_app_priv.login	0
3	29.8.2007	16:36.00	000486	portal30_sso.wwsso_app_admin.ls_login	0
4	29.8.2007	16:36.04	000486	portal30_sso.prijava	0
5	29.8.2007	16:36.11	000486	portal30_sso.preveri_geslo	0
6	29.8.2007	16:36.11	000486	portal30.wwsec_app_priv.process_signon	0
7	29.8.2007	16:36.11	000486	VZ_SKUPNE_IZBERI_SHEMO	0
8	29.8.2007	16:36.19	000486	VZ63_DELAVEC_ZACETNI_PARAMETRI	0
9	29.8.2007	16:36.19	000486	VZ63_DELAVEC_KANDIDATI_PISNEGA_IZPITA	0
10	29.8.2007	16:36.26	000486	VZ63_DELAVEC_VNOS_SPRE_REZULT_IZPITA	0
13	29.8.2007	16:36.31	000486	VZ63_DELAVEC_VNOS_SPRE_REZULT_IZPITA	1
14	29.8.2007	16:36.35	000486	VZ63_DELAVEC_VNOS_SPRE_REZULT_IZPITA	1
15	29.8.2007	16:42.01	000486	VZ63_DELAVEC_VNOS_SPRE_REZULT_IZPITA	1
16	29.8.2007	16:42.07	000486	VZ63_DELAVEC_VNOS_SPRE_REZULT_IZPITA	1
17	29.8.2007	16:42.10	000486	skupne_varnostne.get_porocilo	0
18	29.8.2007	16:43.06	000486	skupne_varnostne.get_porocilo	1
19	29.8.2007	16:43.08	000486	skupne_varnostne.get_porocilo	0
20	29.8.2007	16:43.22	000486	VZ63_LOGOUT	0
22	29.8.2007	16:43.24	000486	PORTAL30.wwsec_app_priv.logout	0
23	29.8.2007	16:43.25	000486	portal30_sso.wwsso_app_admin.ls_logout	0
24	29.8.2007	16:43.25	000486	/logout_fri	0

mesto preko vhodne domače strani in se potem poljubno sprehaja med stranmi, pregleduje izdelke z namenom pregleda ponudbe. Temu uporabniku je podoben tak, ki pride na spletno mesto preko vhodne domače strani in takoj začne z iskanjem določene stvari ali izdelka. Načine prihoda na spletno mesto prikazuje slika 5.6. Zaradi zgoraj navedenih dejstev je veliko težje določiti začetek nove seje pri prepletenih sejah.

Ko uporabnik zapusti spletno trgovino, za sabo v splošnem ne zapusti nobene sledi, ker se mu ni potrebno kakorkoli odjaviti. Spomnimo se, da protokol HTTP nima stanj. Predvidevati moramo, da če uporabnik ne naredi nobene zahteve za spletno stran v nekem časovnem obdobju, je zapustil stran. Pri spletni trgovini smo uporabili časovno prekinitev dolžine 45 min, ki je mnogo manjša od tiste pri sistemu e-Študent. Če se po tem času pojavi nova zahteva s strani iste številke IP, se jo obravnava kot začetek nove seje.

Strežnik spletne trgovine uporablja za zapis podatkov v dnevnik razširjeni format (W3C Extended Log File Format). Vsebuje veliko več atributov kot format CLF, malce se med sabo razlikujejo tudi obstoječi atributi. Glavni atributi formata dnevniške datoteke spletne trgovine so v tabeli 5.5. Med atributi, ki so prisotni v obeh formatih, je razlika v atributu *request*. V razširjenem formatu je ta atribut razdeljen na dva dela: zahtevani vir in parametri zahteve (ang. query parameters), kar zmanjšuje obseg dela in možnost napak pri procesiranju dnevniških podatkov. Trije pomembni dodatni atributi so predhodno obiskana stran (cs-referrer), vsebina piškotka (cs-cookie) in vrsta brskalnika (cs-user-agent). Predhodno obiskana stran pove, na kateri spletni strani je uporabnik kliknil na hiperpovezavo, ki ga je pri-



Slika 5.6: Uporabnikova pot skozi spletno trgovino. Vstop uporabnika na stran globoko v hierarhiji in vstop skozi začetno stran. Povzeto po [43].

peljala na našo stran. Podatek je izjemno pomemben, če hočemo ugotoviti, od kje uporabniki največ prihajajo k nam. Spletna trgovina lahko na ta način spozna svoje potencialne kupce, ugotovi uspešnost oglaševalskih akcij in učinkovitost oglaševanja na drugih spletnih straneh. Piškotek je niz znakov, ki ga spletni strežnik shrani na odjemalčevi strani. Ob vsakem obisku spletnega strežnika odjemalec vedno pošlje strežniku tudi vsebino piškotka. V piškotku se lahko nahajajo podatki, pomembni za strežnik in pozneje za analitike. Spletna trgovina lahko v piškotkih hrani podatke o tem ali je uporabnik prijavljen, katere izdelke je nazadnje gledal, katere izdelke ima v košarici, ipd. Vrsta brskalnika (podpis brskalnika) je atribut, ki ga lahko učinkovito uporabimo v postopku predprocesiranja.

Roboti spletnih iskalnikov se sprehajajo po spletu in ažurirajo svoje baze podatkov. Robot iskalnika preišče celotno spletno mesto v enem ali več časovnih intervalih. Vsi njegovi dostopi so v dnevniku spletnega strežnika vidni kot dostopi navadnih uporabnikov. Ker pa roboti ne odražajo obnašanja uporabnikov, je njihove seje potrebno identificirati in jih zavreči. To lahko storimo s pomočjo prepoznavanja IP naslova ali pa podpisa robota. Vsak lepo vzgojen robot v atributu *cs-user-agent* namesto vrste brskalnika pusti svoj podpis. Primer za iskalnik Google je to Googlebot/2.1. V procesu procesiranja dnevniških datotek moramo samo primerjati imena s tistimi v bazi robotov in njihove seje zavreči. Težava se pojavi v velikem številu robotov, za katere je težko vzdrževati popolno evidenco. Dnevno se pojavljajo novi in novi roboti, kar slabša kakovost podatkov, ker se nam lahko zapisi robotov

izmuznejo med uporabniške seje. Za prepoznavanje robotov smo uporabili javno dostopno zbirko spletnih robotov in jo dopolnili z manjkajočimi, ki so se pojavljali v podatkih.

Tabela 5.5: Format dnevniške datoteke za spletno trgovino.

Atribut	**Opis atributa**
date	datum zahteve za vir strežnika
time	čas zahteve v formatu UTC
cs-username	uporabniško ime identificiranega uporabnika (za anonimne: -)
c-ip	naslov IP odjemalca
cs-method	vrsta zahteve (GET, POST)
cs-uri-stem	zahtevan vir na strežniku (npr. izdelek.aspx)
cs-uri-query	parametri zahteve, če so prisotni (npr. ?id=10&view=1)
sc-bytes	število bajtov, ki jih strežnik pošlje
sc-status	statusna koda HTTP, ki se vrne odjemalcu
cs-host	domensko ime odjemalca
cs-user-agent	vrsta brskalnika na odjemalcu
cs-referrer	predhodno obiskana stran; stran, ki je zagotovila povezavo na to stran
cs-cookie	vsebina prejetega (in poslanega) piškotka
...	drugi atributi: s-port, cs-version, time-taken, s-computername, ...

Prečiščene uporabniške seje spletne trgovine smo dodatno analizirali, da bi ugotovili delež prepletenih uporabniških sej. Predvsem nas je zanimalo, kakšni uporabniki generirajo prepletene seje in ali prepletene seje kaj pogosteje generirajo kupci kot ostali uporabniki. Za osnovo smo vzeli uporabniške seje, kjer smo lahko nedvoumno identificirali uporabnike na podlagi prijave. Pridobili smo podatke o številu čistih sej, številu prepletenih sej in številu kupcev znotraj obeh skupin sej. Rezultati so prikazani v tabeli 5.6. Prvi trije stolpci prikazujejo podatke za čiste seje, drugi trije pa za prepletene seje. Vidimo lahko, da je delež prepletenih sej, ki jih generirajo kupci (30%), skoraj dvakrat več kot tistih, ki jih generirajo ostali uporabniki (17%).

Tabela 5.6: Delež kupcev v prepletenih in neprepletenih sejah.

št. čistih sej	št. kupcev	% kupcev	št. preplet. sej	št. kupcev	% kupcev
10353	1751	17	9343	2792	30

V primerjavi s sistemom e-Študent ima spletna trgovina EnaA veliko več strani. Samih dinamično generiranih spletnih strani je približno dvakrat več kot pri sistemu e-Študent, kar ni tako veliko, vendar sama stran ne določa dovolj dobro prikazane vsebine. Pravo vrednost pri analizi klikotoka predstavlja podatek o tem, kateri izdelek si je uporabnik na strani ogledal. Pod pojmom stran bomo torej imeli v mislih

celoto vira (npr. izdelek.aspx) in dela parametra (npr. ?izdelek_id=2345), ki enolično določa nek izdelek na zahtevani strani. Na ta način se število strani zelo poveča in lahko preseže število 40 000. Vsak dan prihajajo novi izdelki, stari se ukinjajo, kar še pripomore k zapletenosti. Zaradi velikega števila spletnih strani, ki zelo otežujejo analize, smo se odločili strani razvrstiti v skupine. Vsaka skupina vsebuje strani, ki so si v nečem podobne. Na ta način smo dobili nekaj več kot 900 skupin strani.

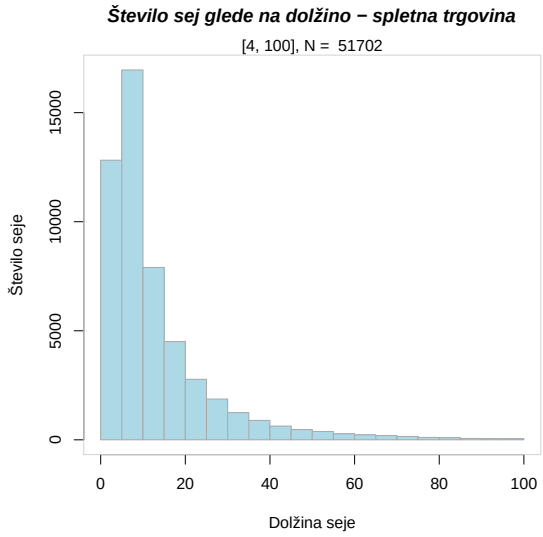
Spletna trgovina ima več obiskov in tudi večje število uporabniških sej. To lahko vidimo iz tabele 5.7, ki prikazuje podatke o številu klikov, številu sej in povprečni dolžini uporabniške seje za 10 dni v mesecu. Spletna trgovina ima v primerjavi s sistemom e-Študent približno petkrat več uporabniških sej. Podobno kot pri podatkih sistema e-Študent smo tudi tu upoštevali samo seje dolžine med 4 in 100. Daljše seje večinoma pripadajo neidentificiranim robotom ali pa so posledica napake v postopku osejevanja. Upoštevane so samo uporabniške seje brez spletnih robotov. Uporabniške seje spletne trgovine so krajše kot pri IS e-Študent. Razliko lahko pojasnimo z načinom dela v spletni aplikaciji, ki se zelo razlikuje od informativnega brskanja po spletnem mestu. Spletna aplikacija lahko zahteva veliko osvežitev neke strani (na kateri se nahaja spletni obrazec), da uporabnik izvede neko željeno nalogo. Upoštevati moramo še prijavni postopek, ki je pri spletni trgovini navzoč le ob dejanskem nakupu.

Tabela 5.7: Število zahtev strani, število sej in povprečna dolžina seje za spletno trgovino.

Dan	Število zahtev	Število sej	Povp. dolžina seje
1	31 125	2 382	13,07
2	46 711	3 450	13,54
3	44 442	3 259	13,64
4	50 117	3 720	13,47
5	78 680	5 446	14,45
6	70 777	4 933	14,35
7	50 798	3 485	14,57
8	56 037	4 135	13,56
9	57 844	4 170	13,87
10	42 220	3 185	13,26

Slika 5.7 prikazuje razporeditev sej glede na dolžino. Vidimo, da je povprečna dolžina seje krajša kot v primeru sistema e-Študent. Število daljših sej je mnogo manjše. Največ sej je dolžine med 4 in 10. Število sej, daljših od povprečja, pa vseeno pada počasneje kot pri sistemu e-Študent.

Podobno kot pri dnevniških datotekah sistema e-Študent smo tudi za spletno trgovino izdelali programe za obdelavo dnevniških zapisov in prenos v področje



Slika 5.7: Histogram števila sej glede na dolžino sej za spletno trgovino. Število vseh sej v vzorcu je 51 702.

za čiščenje in preoblikovanje podatkov. Postopek je podoben tistemu na sliki 5.3. Programi za spletno trgovino so bolj splošno uporabni in bi jih lahko uporabili za katerokoli drugo dnevniško datoteko spletnega strežnika. Dodati pa smo morali modul za identifikacijo in izločanje sej robotov spletnih iskalnikov.

Tabela 5.8 prikazuje primer seje spletne trgovine. Zaradi preglednosti in enostavnosti smo v tabeli prikazali samo najnujnejše atribute. Poleg zaporedne številke zahteve znotraj seje, datuma, časa in zahtevane strani so v tabeli prikazani še parametri zahteve in predhodno obiskana stran (ang. referrer). Seja je dolga deset zahtev. Uporabnik je prišel na spletno stran s strani iskalnika Google, kjer je uporabil iskalni kriterij ”sony tv”. V spletni trgovini se je sprehajal po oddelkih in pregledoval izdelke z določeno šifro. Ob pol enih ponoči je zapustil spletno trgovino, kjer se je zadržal slabih pet minut.

Tabela 5.8: Primer seje v podatkovni bazi za spletno trgovino.

#	datum	čas	ime strani	parametri	predhodna stran
1	1.1.2009	0:24.46	/video/izdelek.asp	dept_id=3286	www.google.si?q=sony+tv
2	1.1.2009	0:24.58	/video/dept.asp	dept_id=3286	-
3	1.1.2009	0:25.16	/video/izdelek.asp	izd_id=812073	-
4	1.1.2009	0:27.09	/video/dept.asp	dept_id=3285	-
5	1.1.2009	0:27.26	/video/dept.asp	dept_id=3287	-
6	1.1.2009	0:27.43	/video/izdelek.asp	izd_id=3287	-
7	1.1.2009	0:28.33	/video/izdelek.asp	izd_id=3287	-
8	1.1.2009	0:29.09	/video/dept.asp	dept_id=3287	-
9	1.1.2009	0:29.23	/video/dept.asp	dept_id=3287	-
10	1.1.2009	0:29.31	/video/dept.asp	dept_id=3287	-

POGLAVJE 6

Eksperimentalni rezultati

6.1 Analiza problemskih situacij

Problem razpletanja sej je bil podrobneje predstavljen v poglavju 3.1. V tem razdelku se bomo posvetili analizi in vrednotenju različnih problemskih situacij. Izbrali bomo različne mejne situacije, pogledali kaj zanje velja in kakšne rezultate dobimo na njihovi osnovi. Postopke in prikaz različnih problemskih situacij smo izvedli na podlagi umetno generiranih podatkov.

6.1.1 Razpletanje vseh možnih prepletov dveh sej

Pri razpletanju sej smo naleteli na vprašanje ali vrstni red prepleta dveh sej vpliva na končni rezultat razpletanja. Če razmislimo, kako deluje markovski model MM1 vidimo, da različeni vrstni red elementov v prepleteni seji vpliva na končen rezultat. Model MM1 je prešibak in zato lahko pri razpletanju pride do dvoumnosti. Ni mogoče zagotoviti, da bi postopek razpletanja vrnil vedno isti dve seji za različne kombinacije elementov. Če je prepletena seja sestavljena iz dveh sej S_1 in S_2, ki si ne delita nobenega skupnega elementa, potem je rezultat razpleta vedno enak, ne glede na različne možnosti prepleta obeh sej. To je posledica dejstva, da v fazi razpletanja nikoli ne pridemo v stanje, kjer se ne bi znali odločiti zaradi dvoumosti. Primer dvoumnosti, na katero lahko naletimo v fazi razpletanja; Imamo dve izvorni seji S_1, S_2 in enega izmed možnih prepletov teh dveh sej S_p:

$$S_1 = [\mathrm{S0\ S6\ S12\ S16\ S15\ S7\ S3\ S18}] \quad \text{in}$$
$$S_2 = [\mathrm{S0\ S6\ S8\ S2\ S15\ S10\ S19}]$$
$$S_\mathrm{p} = [\mathit{S0}\ \mathit{S6}\ \mathit{S12}\ \mathit{S16}\ \mathit{S15}\ \mathit{S0}\ \mathit{S6}\ \mathit{S8}\ \mathit{S2}\ \mathit{S15}\ \mathit{S7}\ \mathit{S10}\ \mathit{S3}\ \mathit{S18}\ \mathit{S19}]$$

Predpostavimo, da imamo markovski model MM1, ki smo ga naučili samo s podatki o prehodih v sejah S_1 in S_2. V postopku razpletanja prepletene seje S_p pridemo do stanja, kot je prikazano na sliki. V prepleteni seji S_p se nahajamo na 11. zaporednem elementu, delno razpleteni seji S_1 in S_2 pa sta v tem trenutku enaki:

$$\begin{aligned} S_{\text{1-delna}} &= [\text{S0 S6 S12 S16 S15}] \\ S_{\text{2-delna}} &= [\text{S0 S6 S8 S2 S15}] \\ S_\mathrm{p} &= [\text{S0 S6 S12 S16 S15 S0 S6 S8 S2 S15 } \underline{\text{S7}} \text{ S10 S3 S18 S19}] \end{aligned} \tag{6.1}$$

Obe seji se trenutno zaključujeta s stanjem $S15$. Pri prirejanju stanja $S7$ smo torej v težavah ker ne vemo, kateri seji naj stanje pripojimo. Imamo 50% možnosti, da naključno izberemo pravo sejo. Če zgrešimo, bodo vsa stanja od $S7$ in $S10$ naprej do konca pripojena napačni seji. Končni rezultat slabo razpletenih sej je:

$$\begin{aligned} S_{\text{1-n}} &= [\text{S0 S6 S12 S16 S15 S10 S19}] \\ S_{\text{2-n}} &= [\text{S0 S6 S8 S2 S15 S7 S3 S18}] \end{aligned}$$

Če bi bil vrstni red v zapleteni seji S_p malce drugačen, do tega scenarija ne bi prišlo. Več kot imata seji S_1 in S_2 skupnih stanj, večja je verjetnost, da pride v fazi razpletanja do napačnih prirejanj in posledično do napačno razpletenih sej. Težavo bi seveda vsaj delno odpravili z bolj zapletenimi stohastičnimi modeli višjih redov kar pa ni predmet razprave ta trenutek. Drugi primer težave pri razpletnju sej, pri različnih kombinacijah prepletov, vidimo na spodnjem primeru. Imamo izvorni seji S_1 in S_2 ter enega izmed možnih prepletov $S_{\mathrm{p}i}$. Model MM1 smo naučili samo s prehodi med stanji v sejah S_1 in S_2.

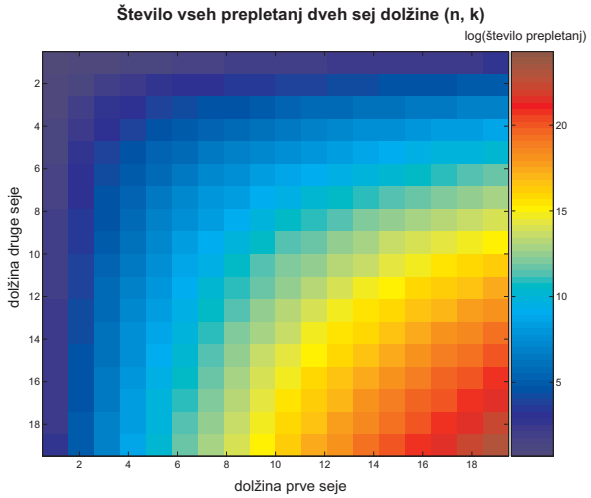
$$\begin{aligned} S_1 &= [\text{S0 S12 S10 S2 S4 S17 S18}] \\ S_2 &= [\text{S0 S14 S12 S13 S2 S10 S18}] \\ S_{\text{pi}} &= [\text{S0 S12 S10 S0 S14 S12 S13 } \underline{\text{S2}} \text{ S10 S2 S18 S4 S17 S18}] \end{aligned}$$

V postopku razpletanja pridemo v prepleteni seji do 8. stanja – $S2$. Delno razpleteni seji S_1 in S_2 sta v tem trenutku enaki $S_1 = [S0\,S12\,S10]$ in $S_2 = [S0\,S14\,S12\,S13]$. Kot je videti iz izvornih sej, bi se moralo stanje $S2$ pripeti prvi seji S_1. Vendar, ker je verjetnost prehoda $S13 \rightarrow S2$ v markovskem modelu očitno večja kot verjetnost prehoda $S10 \rightarrow S2$, se stanje $S2$ doda napačni seji, seji S_2. Po izteku postopka razpletanja dobimo slabo razpleteni seji S_1 in S_2.

$$\begin{aligned} S_1 &= [\text{S0 S12 S10 S18}] \\ S_2 &= [\text{S0 S14 S12 S13 S2 S10 S2 S4 S17 S18}] \end{aligned}$$

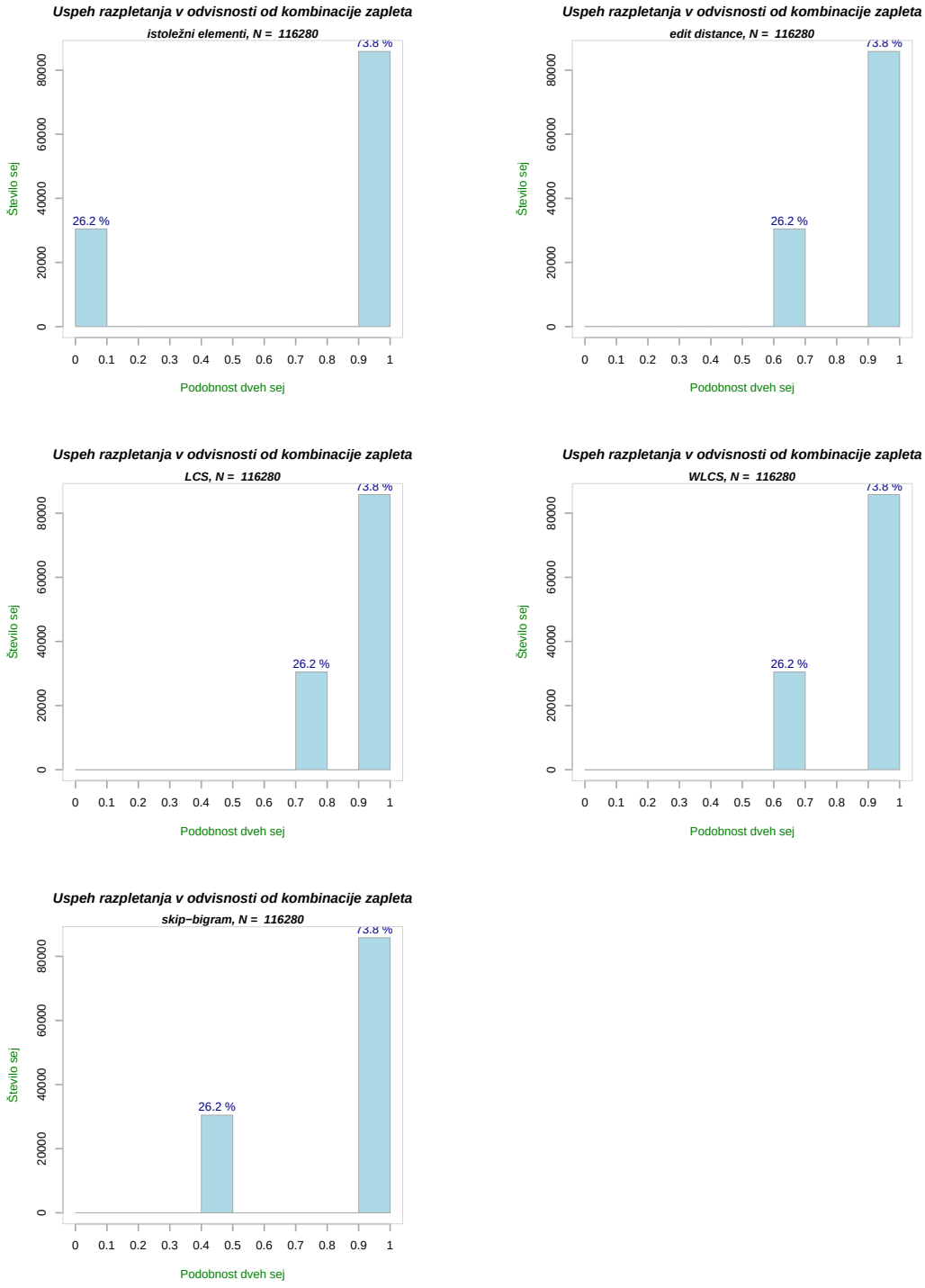
Z namenom videti kako različni prepleti vplivajo na kakovost razpletanja, smo naredili vse kombinacije prepletov dveh sej, jih razpletli in ovrednotili rezultate. Pri izdelavi vseh možnih prepletov dveh sej moramo paziti, da obdržimo vrstni red znotraj prepletene seje za obe seji, ki sestavljata prepleteno sejo. Vrstni red strani v

prepleteni seji mora bit enak vrstnemu redu strani v seji S_1. Podobno velja za sejo S_2. Za prepletanje sej torej ne moremo uporabiti navadnega postopka za izdelavo vseh kombinacij elementov, ker bi zgradili tudi prepletene seje, ki nimajo nič skupnega z izvornimi sejami S_1 in S_2. Izdelati smo morali postopek, ki ohrani izvorni vrstni red. Število vseh možnih prepletov dveh sej je torej manjše od števila vseh kombinacij $\binom{n}{k}$ in sicer je enako $\binom{n1+n2}{n1}$. Podrobnejši opis najdemo v podpoglavju 3.4.2. Število vseh možnih prepletov dveh dej zelo hitro narašča z večanjem dolžine obeh sej. Kakšno je število vseh možnosti pri posameznih vrednostih dolžine lahko vidimo na grafu 6.1.



Slika 6.1: *Temperaturni graf števila vsem možnih kombinacij prepletanj dveh sej. Graf prikazuje število prepletanj za dolžini sej od 1 do 19.*

Želeli smo preveriti kakšne rezultate dobimo, če razpletemo vse možne kombinacije prepletov dveh sej. V skladu z zgornjim razmislekom smo vedeli le, da seje ne bodo vedno pravilno razpletene. Zanimalo pa nas je, kako velik odstotek bo napačno razpletenih in kakšna bo podobnost z dejanskimi. Kakovost razpletenih sej smo ovrednotili z metodami vrednotenja iz prejšnjega podpoglavja: primerjava istoležnih znakov, razdalja med zaporedji, LCS, WLCS in sopojavitev ločenih dvojic elementov. Poskus smo izvedli nad sejami, ki imajo različno število skupnih strani. Seje smo generirali na podlagi grafa spletišča z 20 stanji, najkrajša dolžina seje pa je lahko najmanj 7 strani. Tak primer bomo označili kot **S20D7**, kjer številka za črko *S* pomeni število vseh stanj, številka za črko *D* pa minimalna dolžina seje. Kot smo ugotovili, je kakovost razpletanja odvisna tudi od števila stanj, ki so skupna obem sejam. Poskus smo izvedli za več možnih skupnih stanj. Na sliki 6.2 lahko vidimo rezultate, kjer so obema sejama skupna 3 stanja.



Slika 6.2: *Rezultati razpletanja vseh možnih kombinacij prepletov dveh sej. Na sliki primerjamo rezultate vrednotenja razpletanja po metodah: istoležnih elementov, razdalj med zaporedji, LCS, WLCS in skip-bigram*

6.1.2 Kam padejo tipične prepletene seje

V prejšnjem podpoglavju smo si pogledali, kako vrstni red zapleta dveh sej vpliva na rezultat razpletanja. V realnih sistemih ponavadi nimamo opravka z vsemi možnimi prepleti dveh sej ampak samo z neko manjšo podmnožico. Uporabniki se ponavadi sprehajajo po spletnih straneh po nekem vzorcu, imajo nek način vedenja. Nekateri prepleti so izjemno malo verjetni. Zelo malo verjeten je npr. preplet: $P = [S_1, T_1, S_2, T_2, S_3, T_3, ..., S_n, T_n]$, kjer S_i označujejo strani prve seje, T_i pa strani druge seje. Uporabnik ne preskakuje neprestano med oknoma brskalnika in izvršuje samo po eno operacijo naenkrat. Ponavadi naredi večje število klikov, s katerimi opravi neko zaključeno opravilo. Uporabnik se torej po dveh sprotnih sejah na spletni strani sprehaja po določenem ključu kar pomeni, da za prepletene seje veljajo neke skupne lastnosti. Te so seveda odvisne od konkretnega primera. Prepletene seje spletnih IS so drugačne kot tiste pri spletnih trgovinah. Nikoli ne moremo zagotovo reči, kakšna je neka točno določena uporabniška prepletena seja za nekega uporabnika, kljub temu pa lahko vsaj približno določimo značilnosti *tipične prepletene seje* (TPS). Tipična prepletena seja je določena z dvema parametroma (a, p). Parameter a določa odstotek dolžine prve seje, ki mora preteči, preden se v prepleteni seji pojavi prva stran iz druge seje. Če prevedemo na uporabnika, koliko strani uporabnik zahteva v prvi seji preden odpre drugo sejo in začne istočasno delati v obeh. Drugi parameter p pa določa verjetnost prehoda med sejami. Večja kot je verjetnost prehoda med sejami, krajše bodo podseje, manjša kot bo verjetnost prehoda med sejami, daljše bodo podseje. S pojmom *podseja* označujemo neprekinjeno podzaporedje strani neke seje, ki se nahaja v prepleteni seji. Podsejo lahko pri zaporedjih enačimo z neprekinjenim podzaporedjem. Večje kot so podseje v prepleteni seji, manj je prehodov med sejami.

Naš cilj je bil na realnih podatkih pridobiti vrednost parametrov, ki določajo TPS. V skladu s temi parametri smo naredili preplet določen del tipskih sej. Nato smo prepletene seje po postopku TPS razpletli in ovrednotili rezultate. Naš namen je bil ugotoviti v katero skupino rezultatov na grafu padejo prepleti generirani s pomočjo postopka TPS. Parametre (a, p) smo pridobili iz spletnega IS e-Študent. e-Študent je spletna aplikacija, ki se uporablja že daljše časovno obdobje in je na voljo dovolj podatkov o klikotoku. Sistem omogoča več uporabniških vlog. Vsak uporabnik je lahko član ene ali več uporabniških vlog, ki jo izbere v postopku prijave. Uporabnik je lahko istočasno prijavljen v sistem e-Študent v dveh oknih spletnega brskalnika pod isto ali različno uporabniško vlogo. V dnevniško datoteko spletnega strežnika ni zapisan podatek o seji, zato ne moremo vedno na preprost način ločiti zapisov obeh istočasno trajajočih sej. Če hočemo pridobiti podatke o tipičnih prepletenih sejah moramo iz dejanskega sistema pridobiti prepletene seje, ki jih znamo ločiti. Samo na ta način lahko dobimo dovolj dobre vrednosti za parametra a in p. Za ocenitev parametrov smo uporabili prepletene seje, za katere velja: sestavljene so iz dveh sej; seji pripadata dvem različnim uporabniškim vlogam; lahko jih analiziramo in uspešno ločimo.

Verjetnost prehoda med sejami p lahko izračunamo na več načinov. Lahko jo

izračunamo na podlagi verjetnosti dolžin podsej P. S P_n označimo verjetnost, da je neka podseja dolžine n. V tem primeru velja $(1-p)^n = P_n$, kjer je p verjetnost prehoda med sejami. Na vsakem prehodu v podseji je namreč verjetnost, da se podseja nadaljuje, enaka $(1-p)$. Iz tega sledi, da je $p = 1 - \sqrt[n]{P_n}$. Za končno vrednost p lahko vzamemo maksimalno vrednost ali pa povprečje $\overline{p} = \sum_n p * N / \sum_n N$. Primer izračuna vidimo v tabeli 6.1.

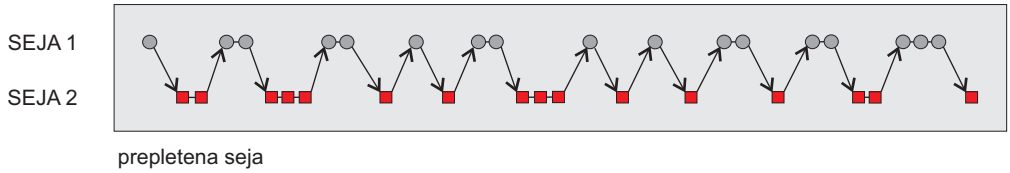
Tabela 6.1: Izračun verjetnosti prehoda med sejami p.

n	**N**	$P = \frac{N}{\sum N}$	**p**
1	10	0,1	0,9
2	20	0,2	0,55
3	30	0,3	0,33
4	40	0,2	0,2
5	50	0,2	0,13
$\sum$	100	1	**0,39**

Drugi način prikazuje enačba (6.2). Verjetnost prehoda med sejami je kvocient med številom vseh prehodov med sejami in vsoto dolžin vseh sej. V enačbi (6.2) n predstavlja število vseh dolžin podsej, i pa dolžino podseje.

$$p = \frac{\text{št. prehodov}}{\sum \text{dolžina seje}} = \frac{\sum \text{št. sej} - 1}{\sum_{i=1}^{n} (\text{št. sej})_i * i} \tag{6.2}$$

Za boljše razumevanje izračunajmo verjetnost prehoda med sejami p na podlagi prepletene seje na sliki 6.3. Prepletena seja je sestavljena iz desetih sej dolžine ena, sedmih sej dolžine dve in treh sej dolžine tri. Število prehodov med sejami je 19, kar je enako številu vseh sej minus ena $(10+7+3)-1 = 19$. Vsota dolžine sej pa je enaka $10*1+7*2+3*3 = 33$. Rezultat je torej $p = \frac{19}{33} = 0.58$.



Slika 6.3: *Prepletena seja. Slika prikazuje način izračuna verjetnosti prehoda med dvema sejama.*

Uporabili smo drugi način za izračun verjetnosti prehoda med sejami. Na podlagi 163 testnih prepletenih sej, smo prišli do rezultatov za oba parametra: $a = 0.43$ in $p = 0.199$. Parameter a pove, da ima uporabnik $\approx$40% prve seje aktivno samo to sejo. Šele nato odpre novo sejo, ki se prepleta s prvo sejo. Parameter p pove verjetnost, da bo naslednja uporabnikova stran pripadala drugi seji kot zadnja zahtevana.

Manjša verjetnost p pomeni večje dolžine podsej. Izračunali smo tudi povprečno dolžino seje, ki je za primer sistema e-Študent ≈ 15 strani.

Na podlagi parametrov, ki smo jih pridobili iz dejanskega sistema lahko generiramo tipične prepletene seje uporabnikov. Zanima nas, kakšna je povezava med rezultati razpleta tipičnih prepletenih sej in rezultati razpleta vseh možnih prepletenih sej. Podobno kot pri primeru vseh prepletenih sej smo najprej izbrali dve seji in tvorili vse možnosti prepletov teh dveh sej. Izbrani seji, s pomočjo katerih smo tvorlili vse možne preplete, smo potem prepletli po postopku TPS. Postopek prepletanja teh dveh sej smo večkrat ponovili. Na sliki 6.4 vidimo rezultate razpletanja za različne metode vrednotenja. Na primeru sta uporabljeni seji: $S_1 = [\mathrm{S0\,S13\,S15\,S7\,S3\,S16\,S5\,S18}]$ in $S_2 = [\mathrm{S0\,S2\,S8\,S10\,S3\,S14\,S18}]$. Moder stolpec predstavlja vse možne preplete, rdeč stolpec pa preplete po postopku TPS.

6.1.3 Rezultati razpletanja sej, podobnih realnim

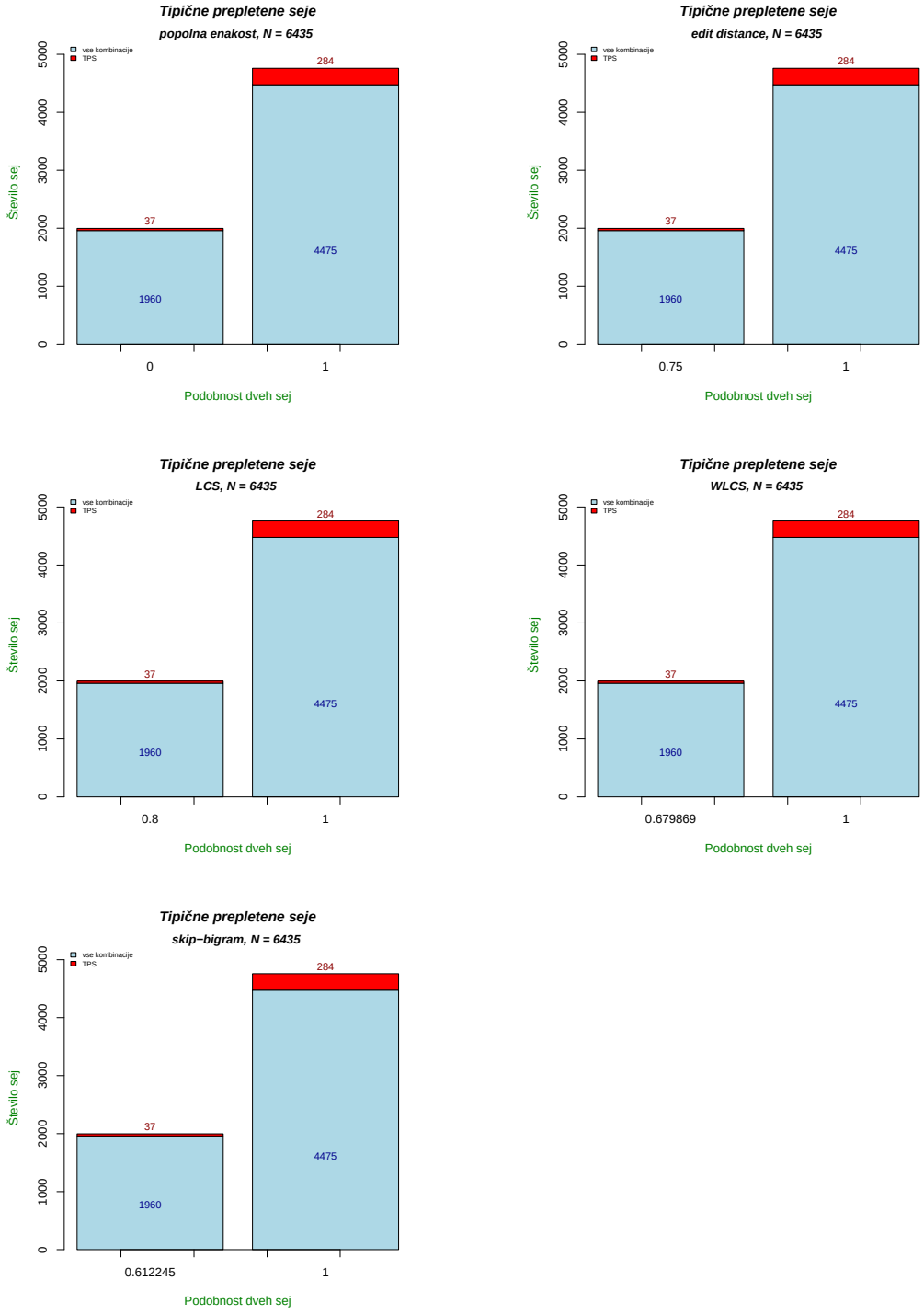
V prejšnjih dveh primerih smo ugotavljali, kako je razplet dveh sej odvisen od različnih prepletov teh dveh sej. Pogledali smo si tudi kako se razpletajo seje, ki so podobne realnim prepletenim sejam glede na različne preplete dveh sej. V tem poglavju si bomo pogledali kakšne rezultate dobimo pri razpletanju tipičnih prepletenih sej, to je tistih, ki so podobne prepletenim sejam v realnem sistemu. Za test smo uporabili testni model spletišča, ki ima 20 stanj. Na podlagi načrta strani smo generirali seje dolžine vsaj 8 strani (S8G20). Kreirali smo 80 tipskih sej. Uporabili smo gaussovo porazdelitev za razporeditev števila uporabnikov po tipskih sejah in generiranje učne množice podatkov. Največje število uporabnikov za sejo je 150. Testno monožico podatkov so predstavljale vse kombinacije 80 tipskih sej. Prepletene seje smo izdelali s postopkom, ki nam generira tipične prepletene seje. Vrednosti parametrov a in p sta enaki kot v prejšnjem primeru. Razpletene seje smo ovrednotili z vsemi petimi metodami vrednotenja. Rezultate lahko vidimo na sliki 6.5 in v tabeli 6.2.

Tabela 6.2: Rezultati razpletanja sej, generiranih po postopku TPS

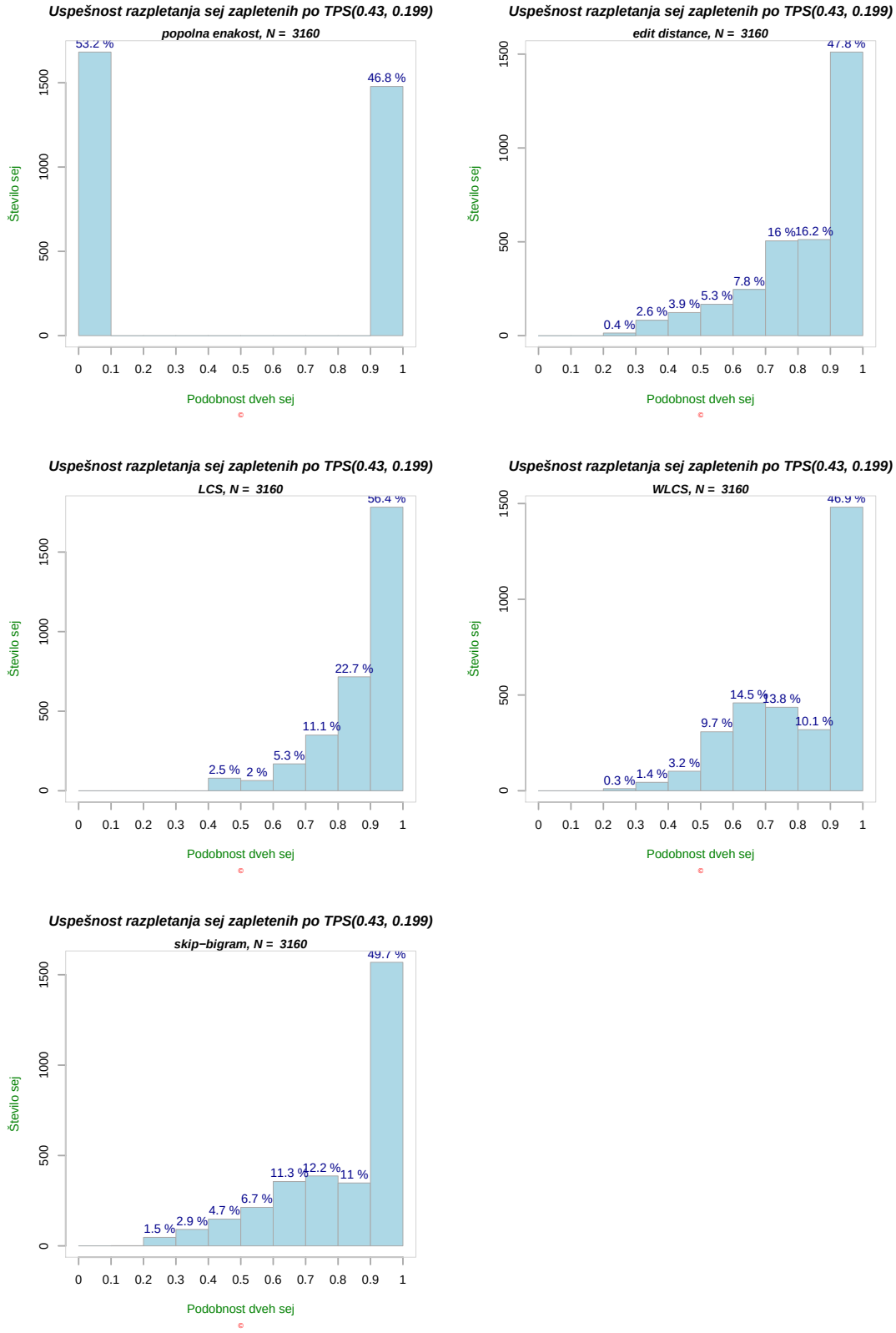
Metoda				
popolno ujemanje	razdalja med zaporedji	LCS	WLCS	skip-bigram
0,468	0,851	0,894	0,825	0,828

6.1.4 Rezultati particioniranja seje v dve ločeni seji

V podpoglavju 3.4.4 smo predstavili Stirlingova števila drugega reda in naredili povezavo z problemom razpletanja sej. Ugotovili smo, da lahko sejo dolžine n razdelimo v dve neprazni množici na $S(n,2)$ načinov. Povedano drugače, sejo dolžine n lahko razpletemo na dve seji S_1 in S_2 na natančno $S(n,2)$ načinov. Izmed vseh



Slika 6.4: *Kam med vse preplete padejo tipične prepletene seje. Rezultati razpletanja so vrednoteni po vseh metodah.*



Slika 6.5: *Rezultati razpleta prepletenih sej, ki so podobne tistim v dejanskem sistemu.*

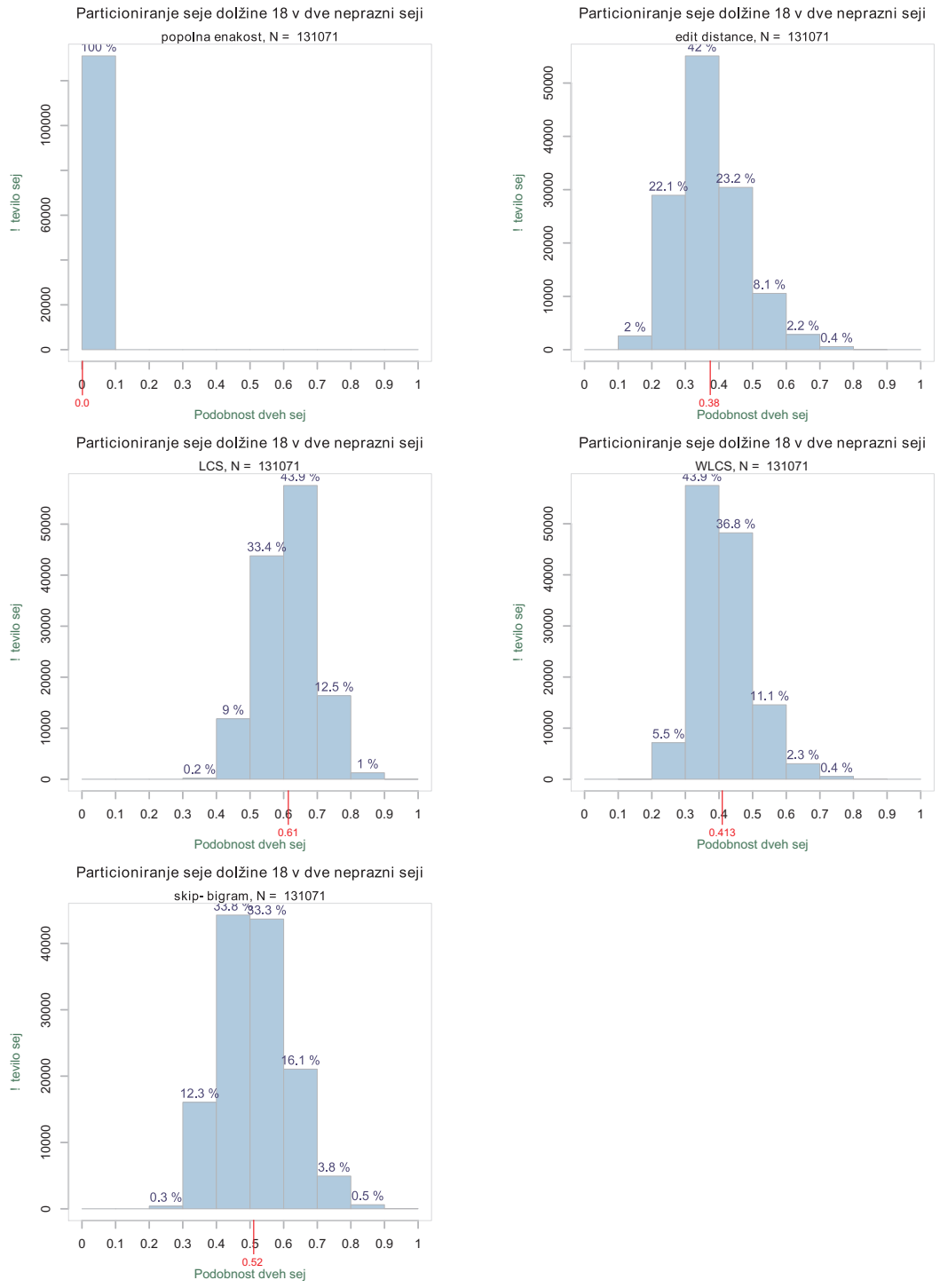
možnih razpletov je seveda samo eden popolnoma pravilen. S stališča čimboljšega razpletanja nam podatek ne pove veliko. Pove pa nam kako majhna možnost je, da z naključnim razpletanjem dobimo popolnoma pravilno razpleteni seji. Ostale možnosti so samo bolj ali manj podobne popolnoma pravilnemu razpletu. Zanima nas, kakšen je rezultat podobnosti za vse možnosti razpletov prepletene seje dolžine n. Rezultat je odvisen od metode vrednotenja, zato smo razplete vrednotili z različnimi metodami. V testnem primeru smo uporabili prepleteno sejo dolžine 18 kar pomeni, da je število vseh razpletov 131071. Rezultate vidimo na sliki 6.6.

Množica razpletenih sej je v vseh primerih enaka. Rezultati so odvisni od uporabljene metode vrednotenja. Najbolj nepopustljiva je metoda vrednotenja, kjer nas zanimajo samo pravilni razpleti. Pravilen razplet je samo eden, zato je povprečena podobnost enaka nič. Ostale metode vrednotenja nagrajujejo tudi podobnost med posameznimi deli zaporedja. Vsaka metoda drugače razumeva podobnost med zaporedji. Po povprečnih vrednostih podobnosti si metode sledijo: metoda razdalj med dvemi zaporedji, metoda uteženega najdaljšega skupnega podzaporedja, metoda statistike sopojavitev ločenih dvojic elementov (ang. skip-bigram) in metoda najdaljšega skupnega podzaporedja. Ni mogoče zagotovo reči katera metoda je boljša in katera ne. Zelo je odvisno od tega, kateri elementi podobnosti nas zanimajo in kaj za nas pomeni, da je eno zaporedje bolj kot drugo podobno ciljnemu zaporedju.

6.1.5 Rezultati glede na naključne razplete

Markovski model prvega reda so se izkazali kot primerni za modeliranje stohastičnih procesov in so se izkazali kot primerni za modeliranje in napovedovanje obnašanja uporabnikov na spletnih straneh. Kljub temu pa imajo določene omejitve. V primerjavi z markovskimi modeli višjih redov imajo ponavadi manjšo natančnost napovedovanja obnašanja uporabnikov [72]. Razlog tiči v dejstvu, da ti modeli ne posegajo dovolj v preteklost, da bi razločili med različnimi vzorci uporabe. Za testne primere v tem poglavju smo uporabili enostaven markovski model prvega reda. Kot smo videli v prejšnjih primerih z njim lahko razpletamo seje z neko uspešnostjo. Zanima nas, kako uspešni smo sploh pri razpletanju in ali se markovski model prvega reda izkaže kot dovolj dobra alternativa. Preverili smo, kako se model MM1 odreže v primerjavi z naključnim razpletanjem sej. To je preprost postopek, kjer naključno dodelimo strani prepletene seje v dve razpleteni seji. Od naključnega razpletanja ne pričakujemo posebej dobrih rezultatov. Zelo majhna možnost je, da pravilno razpletemo obe seji. Različne metode vrednotenja bodo sicer bolje ali slabše nagradile kakovost razpleta. S primerjavo med naključnim razpletanjem in razpletanjem po modelu MM1 smo hoteli ugotoviti ali so in koliko so rezultati razpletanja z uporabo MM1 boljši. Razlika med rezultati mora biti dovolj velika, da lahko trdimo, da je ena metoda boljša od druge.

Rezultate smo preverili z uporabo *Z-testa*. Z njim preverimo ali so rezultati določenega vzorca (testni primer) zunaj ali znotraj področja nekega drugega vzorca



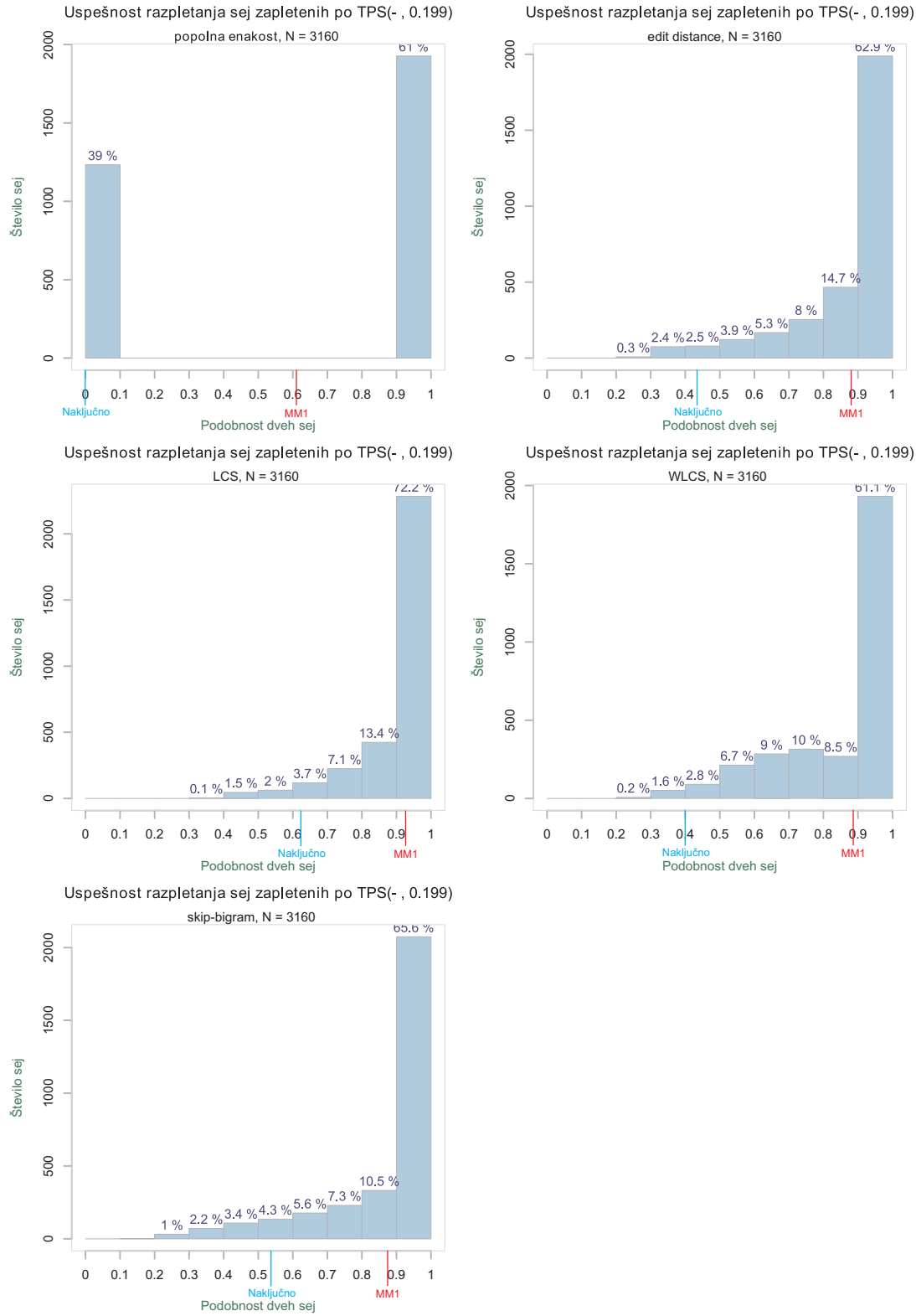
Slika 6.6: *Rezultat vrednotenja razpletanja vseh možnih razpletanj seje dolžine 18.*

(populacija). Za izvedbo Z-testa potrebujemo povprečje populacije in standardno deviacijo populacije $N(\mu,\sigma)$. Potrebujemo pa tudi povprečje testnega primera $\bar{x}$. S pomočjo teh vrednosti lahko izračunamo rezultat *Z-testa*. Način izračuna posameznih vrednosti prikazuje enačba (6.3). Rezultat *Z-testa* je razdalja med popvrečjem populacije μ in povprečjem testnega primera $\bar{x}$, izražena v enotah standardne deviacije populacije σ. To ogrodje lahko uporabimo tudi za primerjanje rezultatov naših dveh metod. V našem primeru populacijo predstavljajo rezultati, dobljeni z naključnim razpletanjem, testni primer pa rezultati dobljeni z metodo MM1.

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n}, \qquad \sigma = \sqrt[2]{\frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n}}, \qquad Z_i = \frac{x_i - \mu}{\sigma_{pop}} \tag{6.3}$$

Test smo izvedli nad primerom $G8S20$. Kreirali smo 80 tipskih sej. Učno množico za model MM1 so tvorile po gaussu porazdeljene uporabniške seje po tipskih sejah. Največje število uporabnikov na tipsko sejo je 150. Testne podatke smo kreirali s pomočjo prepletanja vseh možnih kombinacij tipskih sej po postopku TPS(-,p). Vse prepletene seje smo nato razpletli po postopku *naključnega* razpletanja in z uporabo *modela MM1*. Vsako prepleteno sejo smo po naključnem postopku večkrat razpletli, da smo dobili več podatkov. Graf, ki smo ga dobili pri vrednotenju kakovosti razpletanja po metodi naključnega razpletanja je zelo podoben grafu, ki smo ga dobili s particioniranjem seje dolžine n v dve neprazni podmnožici (Stirlingova množica $S(n,2)$). Rezultate razpletanja smo vrednotili po metodah: popolno ujemanje, razdalja med zaporedji, najdaljše skupno podzaporedje, uteženo skupno podzaporedje in z analizo sopojavitev ločenih dvojic elementov (ang. skip bigram). Rezultate po posameznih metodah vidimo in primerjamo na sliki 6.7.

Grafi prikazujejo rezultate razpletanja z metodo MM1. Na osi Y je podobnosti med razpletenimi sejami in sejami, ki sestavlajo prepleteno sejo. Rezultat podobnosti je odvisen od uporabljene metode vrednotenja. Na vseh grafih je na osi X z modro barvo označeno mesto, ki predstavlja povprečno vrednost podobnosti razpletanja z uporabo metode naključnega razpletanja seje. Z rdečo barvo pa je označena povprečna podobnost razpletenih sej, dobljena z uporabo modela MM1. Ne glede na uporabljeno metodo vrednotenja lahko vidimo, da daje model MM1 boljše rezultate kot jih dobimo pri naključnem razpletanju. Pri nekaterih metodah vrednotenja so razlike med povprečnimi vrednostimi večje, pri drugih manjše, kar je rezultat načina vrednotenja rezultatov. Pri popolnem ujemanju je razlika med metodama največja. Zelo težko je namreč z naključnim razpletanjem popolnoma točno razplesti sejo. Z modelom MM1 pa nam je v tem primeru uspelo popolnoma točno razplesti kar velik del sej. Pri metodi vrednotenja na podlagi najdaljšega skupnega podzaporedja je razmak manjši. To je zaradi tega, ker ta metoda močneje nagrajuje pravilne dele podsej. Številske vrednosti in rezultate *Z-testa* lahko vidimo v tabeli 6.3.



Slika 6.7: *Primerjava kakovosti rezultatov MM1 v primerjavi z naključnim razpletanjem.*

Tabela 6.3: Primerjava uspešnosti modela MM1 glede na naključno razpletanje

		TF	Levenshtein	LCS	WLCS	skip-bigram
Naključno	μ	0,0	0,43	0,62	0,398	0,53
	σ	0,005	0,11	0,083	0,079	0,13
MM1	$\bar{x}$	0,61	0,89	0,927	0,87	0,89
	σ_x	0,488	0,169	0,123	0,18	0,18
	$\mathbf{Z}$	114,3	4,16	3,69	6,0	2,68

6.2 Rezultati razpletanja realnih podatkov

Razvite postopke, ki smo jih predstavili v prejšnjem poglavju, smo preverili na dveh virih podatkov o klikotoku: spletni trgovini EnaA in spletnem študijskem IS e-Študent. Vira podatkov o klikotoku sta zelo različna, kar nam je bilo v pomoč pri testiranju, ali so razviti postopki splošnonamenski in jih lahko uporabimo na prepletenih sejah poljubnega vira klikotoka. Podatke iz obeh virov klikotoka smo očistili in pripravili tako, kot smo predstavili v poglavju 5. Prečiščene celovite podatke o klikotoku obeh virov smo analizirali in ugotovili kakšne so tipične uporabniške seje za oba vira klikotoka. V postopkih prepletanja in razpletanja smo uporabili samo tiste podatke, ki ustrezajo tipičnim (običajnim) uporabniškim sejam.

Pri razpletanju smo pri obeh metodah uporabili naučen markovski model prvega reda. Za učenje markovskega modela smo uporabili čiste seje iz spletnega podatkovnega skladišča. Določen del podatkov iz spletnega podatkovnega skladišča smo namenili za učno in testno množico. 70% čistih sej iz te množice je tvorilo učno množico za markovski model, 30% čistih sej iz te množice pa smo uporabili za izdelavo testne množice prepletenih sej. Tabela 6.4 prikazuje velikost učne množice, število čistih sej za generiranje testne množice in velikost testne množice prepletenih sej za oba vira podatkov.

Tabela 6.4: Velikost učne in testne množice za oba vira podatkov.

vir	učna množica	čiste seje za testno množico	umetno prepl. seje
e-Študent	113 957	48 840	24 400
EnaA	35 517	15 221	7 600

Postopke razpletanja smo izvedli na umetno generiranih prepletenih sejah iz dejanskih, uporabniških čistih sej. Pri tem smo uporabili podatke o zgradbi tipične prepletene seje. Tako smo lahko generirali prepletene seje, ki so čimbolj podobne dejanskim prepletenim sejam. Prednost generiranih prepletenih sej v primerjavi z dejanskimi je v tem, da imamo pri generiranih prepletih podatke o tem, iz katerih sestavnih sej je preplet sestavljen. Na ta način lahko po končanem razpletanju vrednotimo pravilnost razpletanja in sklepamo na kakovost postopkov za razpletanje. Pri generiranju prepletenih sej smo uporabili čiste seje dolžine med 4 in 100 uporabniškimi zahtevami. Generirane prepletene seje lahko vsebujejo različno število sestavnih čistih sej. Na podlagi analize podatkov spletnega študijskega IS smo ugotovili, da je največ prepletenih sej, kjer se prepletata dve oz. tri uporabniške seje. Sej z večimi prepleti je občutno manj, zato smo jih zanemarili. Generirali in razpletali smo torej prepletene seje, ki vsebujejo eno, dve ali tri sestavne čiste seje. Število prepletov z različnimi števili sestavnih sej je enakomerno razporejeno. Tretjina je torej sej z eno sestavno sejo (dejansko je to neprepletena seja), tretjina takih z dvema prepletajočima se elementarnima sejama in tretjina s tremi prepletajočimi se sejami.

6.3 Validacijska množica

Pred razpletanjem sej smo najprej določili vrednost parametrov metod. Pri obeh metodah je bilo potrebno določiti parameter ω. Faktor ω vpliva na verjetnost, da je neko stanje znotraj prepletene seje določeno kot začetno stanje ene izmed razpletenih sej. Pogosto je verjetnosti začetka nove seje, pridobljena iz markovskega modela, prevelika. To popravimo z ustrezno vrednostjo faktorja ω. Podrobneje smo ga predstavili v poglavju 4.7.

6.3.1 Razpletanje z markovskim modelom

Faktor ω smo določili s pomočjo rezultatov razpletanja na validacijski množici. Za vsak vir podatkov o klikotoku in metodo razpletanja smo izdelali validacijsko množico velikosti 1500 prepletenih sej in izvedli razpletanje. Razpletanje smo ponovili pri sedmih različnih vrednostih parametra ω. Rezultate smo podobno kot pri razpletanju testne množice ovrednostili na podlagi vseh petih metod vrednotenja, ki smo jih predstavili v razdelku 4.5. Na podlagi rezultatov smo določili najprimernejšo vrednost ω. Rezultati razpletanja na validacijski množici z uporabo markovskega modela za sistem e-Študent in spletno trgovino so prikazani v tabeli 6.5. Prvi del tabele vsebuje rezultate za sistem e-Študent, drugi del tabele pa rezultate za spletno trgovino EnaA. Vsaka vrstica tabele predstavlja rezultate razpletanja pri določeni vrednosti ω. Vrednosti v tabeli predstavljajo povprečno podobnost na podlagi mere-F med razpletenimi in dejanskimi sestavnimi sejami. Vrednosti so lahko med 0 (popolna različnost) in 1 (popolna enakost). Stolpci tabele predstavljajo različne metode vrednotenja.

Drugi stolpec tabele 6.5 vsebuje vrednotenje razpletov po načelu popolne enakosti. Seje so lahko ali popolnoma pravilno razpletene (vrednost 1) ali pa napačno razpletene (vrednost 0). Napačen razplet po tej metodi pomeni različen položaj že ene same strani v eni izmed razpletenih sej glede na sestavne seje. Pri tej metodi povprečna podobnost predstavlja kar delež vseh popolnoma pravilno razpletenih sej. Rezultat $0,43$ v prvi vrstici in drugem stolpcu torej pomeni, da je bilo popolnoma pravilno razpletenih na sestavne dele kar 43% vseh prepletenih sej.

Pri drugih metodah vrednotenja podobnost med razpletenimi in sestavnimi sejami za posamezen razplet zavzema tudi vrednosti med 0 in 1, zato ne moremo sklepati na podoben način kot pri metodi popolne enakosti. Postopek izračuna podobnosti med dvema razpletoma po posamezni metodi smo opisali v razdelku 4.5.6.

Iz tabele 6.5 vidimo, da dobimo najboljše rezultate razpletanja pri vrednosti faktorja $\omega = 1$. To velja za oba vira podatkov, ne glede na izbrano metodo vrednotenja podobnosti. Tako pri večanju kot manjšanju vrednosti ω se rezultati poslabšajo. Upad števila razpletenih sej v odvisnosti od ω se pri obeh virih podatkov razlikuje. Razlog lahko najdemo v bolj ali manj dobro definiranih začetnih straneh pri obeh virih podatkov. Večji upad je pri sistemu e-Študent, saj se število pravilno razpletenih sej pri $\omega = 0,0001$ izenači za oba vira podatkov. Sklepamo, da je boljši rezultat pri

Tabela 6.5: Rezultati razpletanja z uporabo markovskega modela na validacijski množici podatkov. Vrednosti predstavljajo povprečno podobnost med razpletenimi in dejanskimi sestavnimi sejami.

e-Študent					
ω	popolna enakost	edit-distance	LCS	WLCS	skip-bigram
10	0,430	0,625	0,639	0,611	0,614
1	0,477	0,732	0,754	0,717	0,716
0,1	0,406	0,633	0,671	0,631	0,612
0,01	0,359	0,529	0,573	0,538	0,507
0,001	0,332	0,498	0,544	0,509	0,474
0,0001	0,333	0,497	0,542	0,507	0,473
0,00001	0,333	0,484	0,526	0,493	0,462
EnaA					
ω	popolna enakost	edit-distance	LCS	WLCS	skip-bigram
10	0,250	0,305	0,313	0,305	0,299
1	0,351	0,517	0,545	0,520	0,497
0,1	0,336	0,412	0,427	0,413	0,402
0,01	0,335	0,409	0,424	0,411	0,400
0,001	0,337	0,411	0,426	0,413	0,402
0,0001	0,339	0,411	0,426	0,413	0,402
0,00001	0,338	0,409	0,424	0,411	0,400

sistemu e-Študent posledica prav zmožnosti bolj točnega določanja začetnih stanj razpletenih sej.

Vrednotenje glede na število sej v prepletu

Tabela 6.6 prikazuje rezultate razpleta z uporabo MM glede na število vsebovanih elementarnih sej v prepletu. Uporabljeno je vrednotenje na podlagi popolne enakosti. Prvi del tabele vsebuje podatke za sistem e-Študent, drugi del pa za spletno trgovino. Vsaka vrstica pripada skupini sej z določenim številom vsebovanih prepletov. Razvidno je, da so rezultati metode neposredno in tesno povezani s faktorjem ω. Če povečamo ω, se to zelo hitro odrazi na rezultatih. Pri vrednosti faktorja $\omega = 1$ metoda dobro razpleta vse skupine prepletenih sej. Z manjšanjem vrednosti faktorja ω (manjšanjem verjetnosti, da je neko stanje začetno stanje) sicer pridobimo na točnosti razpletanja dejansko neprepletenih sej, vendar se hitro slabša rezultat za ostali dve skupini prepletov, ki vsebujejo dve ali tri prepletajoče se elementarne seje. Pri vrednostih $\omega \leq 0,001$ se ni pravilno razpletla niti ena izmed 500 prepletenih sej s tremi prepletajočimi se sejami.

Slika 6.8 prikazuje rezultate razpletanja z uporabo MM v odvisnosti od faktorja ω. Prvi graf pripada sistemu e-Študent, drugi pa spletni trgovini. Rezultati so vre-

Tabela 6.6: Rezultati postopka razpletanja z uporabo MM na validacijski množici glede na število vsebovanih elementarnih sej.

e-Študent							
št. elem. sej	ω						
	10	1	0,1	0,01	0,001	0,0001	0,00001
1	0,782	0,872	0,982	0,994	0,994	0,998	0,998
2	0,374	0,430	0,222	0,084	0,004	0,004	0,004
3	0,140	0,140	0,018	0	0	0	0
EnaA							
št. elem. sej	ω						
	10	1	0,1	0,01	0,001	0,0001	0,00001
1	0,532	0,808	0,952	0,954	0,956	0,960	0,962
2	0,190	0,216	0,056	0,052	0,056	0,056	0,052
3	0,028	0,032	0	0	0	0	0

dnoteni po metodi popolnega ujemanja sej (100% pravilnost ali napačnost). Iz grafa je razvidno, da je metoda najbolj uravnotežena in najbolje razpleta pri vrednosti faktorja $\omega = 1$. V tem primeru dobro razpleta vse skupine prepletenih sej.

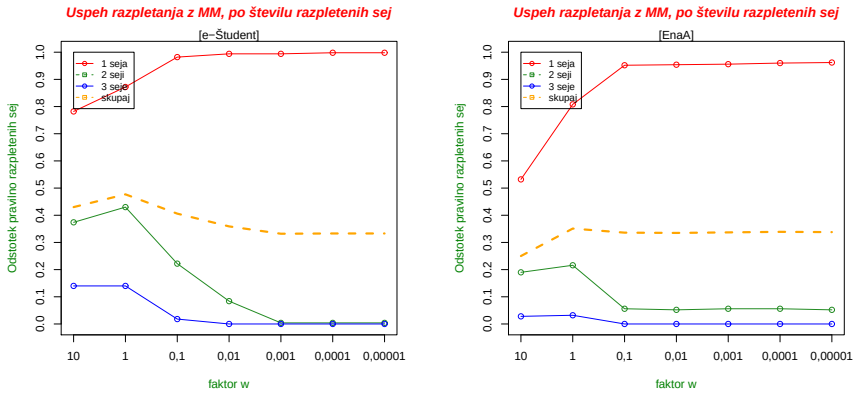
6.3.2 Razpletanje z uporabo RBFS

Validacijsko množico 1500 prepletenih sej smo razpletli tudi z uporabo postopka RBFS. Rezultati se nahajajo v tabeli 6.7. Prvi del tabele vsebuje podatke za sistem e-Študent, drugi del pa za spletno trgovino. Struktura tabele in pomen atributov je enak kot pri tabeli 6.5. Pri uporabi postopka RBFS dobimo najboljše rezultate za oba vira podatkov pri vrednosti $\omega = 0,001$. To pomeni, da pri uporabi te metode razpletanja dajemo preveliko težo verjetnosti, da je izbrano stanje začetno stanje nove seje. Najenostavneje je to vidno, če se osredotočimo na vrednotenje na podlagi popolne enakosti. Za oba vira podatkov velja, da uspešnost razpletanja narašča z zmanjševanjem pomena začetnega stanja (utež ω) do vrednosti $\omega = 0,001$, ko dosežemo najboljše rezultate. Z nadaljnim manjšanjem uteži ω se rezultati razpletanja zopet slabšajo.

Na podlagi zgornjih rezultatov smo pri razpletanju z uporabo markovskega modela določili utež $\omega = 1$. Pri uporabi postopka RBFS pa smo za utež ω določili vrednost $0,001$. Faktor ω je neodvisen od vira podatkov.

6.4 Rezultati razpletanja pri uporabi MM

Prepletene seje iz obeh virov podatkov smo najprej razpletli s pomočjo prve metode, ki za razpletanje uporablja markovski model. Za oba vira podatkov smo obdelali in



Slika 6.8: *Uspeh razpletanja v odvisnosti od uteži pomena začetnega stanja ω.*

razpletli 3000 prepletenih sej. Množica prepletenih sej je bila sestavljena iz različnega števila prepletajočih se elementarnih sej. Tretjina prepletenih sej je vsebovala prepletajoči se dve seji, druga tretjina prepletajoče se tri seje, ostanek pa so tvorile neprepletene seje (prepleti z eno samo elementarno sejo). Razpletene seje smo primerjali s sestavnimi sejami in primerjali medsebojno podobnost. Za vrednotenje podobnosti sej smo uporabili 5 različnih metod vrednotenja, ki smo jih podrobneje predstavili v podpoglavju 4.5. Uporabili smo utež $\omega = 1$.

Rezultati razpletanja z uporabo markovskega modela za oba vira podatkov so prikazani v tabeli 6.8. Prva vrstica tabele vsebuje rezultate za sistem e-Študent, druga vrstica tabele pa rezultate za spletno trgovino EnaA. Vsak stolpec tabele pripada eni izmed metod vrednotenja rezultatov.

Pri razpletanju z uporabo MM dobimo boljše rezultate pri sistemu e-Študent. Popolnoma pravilno se razplete 46% prepletenih sej sistema e-Študent in 34% sej spletne trgovine. Tudi če pogledamo ostale metode vrednotenja vidimo, da je razmerje približno enako kot pri vrednotenju na podlagi popolne enakosti. To pomeni, da nimamo scenarija, kjer bi bilo pri spletni trgovini sicer več sej napačno razpletenih, vendar bi imeli precej več razpletov z zelo visoko stopnjo podobnosti. Uporabniške seje pri sistemu e-Študent imajo bolj jasno definirano strukturo, lažje je določiti začetne strani elementarnih sej. To se zrcali tudi na boljših rezultatih razpletanja.

Sliki 6.9 in 6.10 prikazujeta grafe metod vrednotenja rezultatov razpletanja prepletenih sej z uporabo MM pri vrednosti faktorja $\omega = 1$. Ovrednoteni rezultati za sistem e-Študent so prikazani na sliki 6.9, za spletno trgovino EnaA pa na sliki 6.10. Vsak graf na sliki se nanaša na eno metodo vrednotenja. Ime metode vrednotenja se nahaja pod naslovom grafa. Os X predstavlja intervale za podobnost sej na osnovi mere-F, os Y pa predstavlja število razpletenih sej, ki padejo v določen interval podobnosti. Višina stolpcev predstavlja število sej, ki padejo v določen interval podobnosti. Nad stolpcem je podatek o odstotku vseh sej. Prekinjana črta grafa

Tabela 6.7: Rezultati razpletanja prepletenih sej na validacijski množici z uporabo postopka RBFS. Vrednosti predstavljajo povprečno podobnost med razpletenimi in dejanskimi sestavnimi sejami.

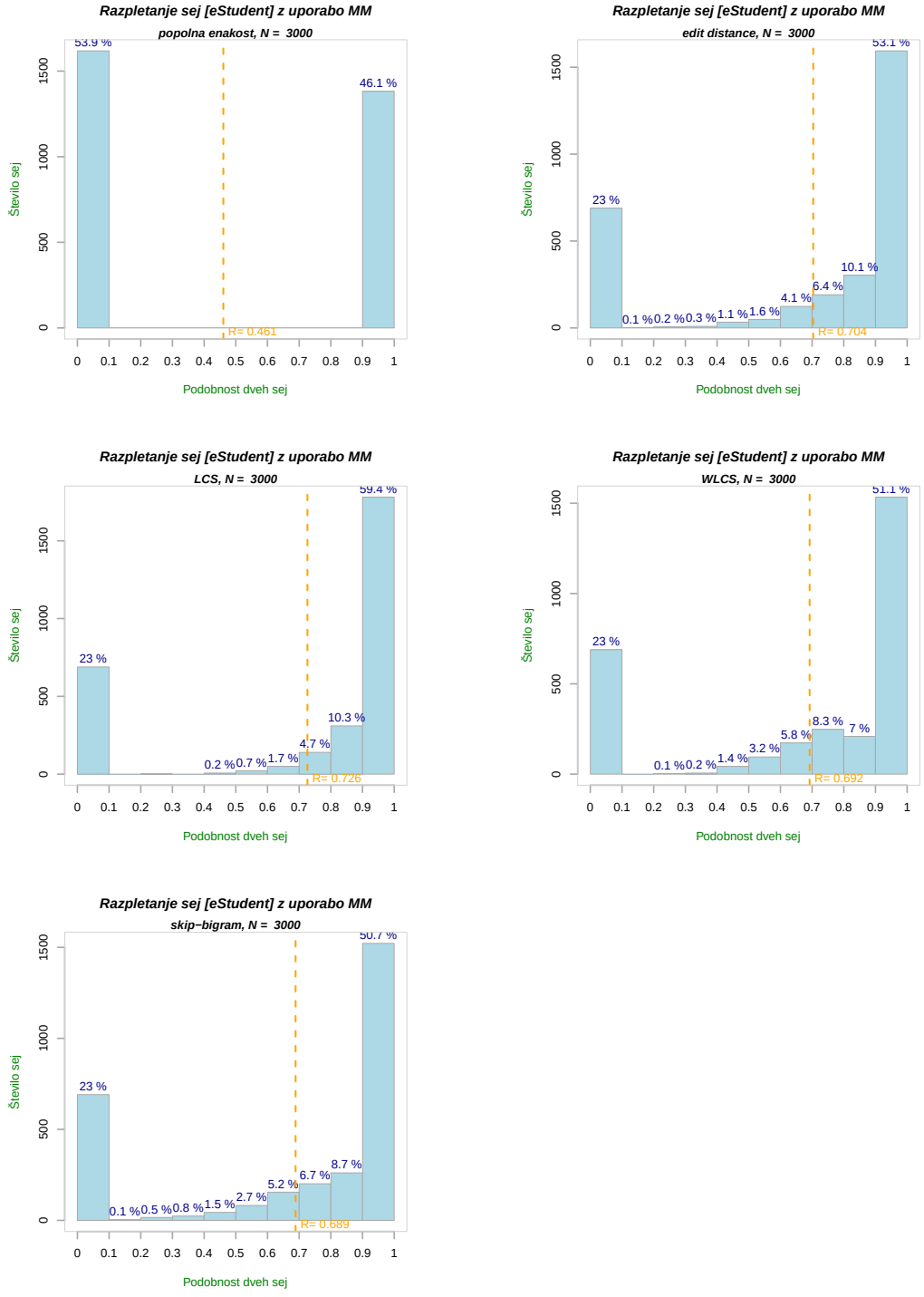
e-Študent					
ω	popolna enakost	edit-distance	LCS	WLCS	skip-bigram
10	0,401	0,547	0,565	0,549	0,532
1	0,458	0,648	0,673	0,652	0,626
0,1	0,509	0,736	0,769	0,743	0,709
0,01	0,521	0,760	0,796	0,769	0,730
0,001	0,528	0,766	0,803	0,775	0,734
0,0001	0,528	0,756	0,791	0,765	0,728
0,00001	0,520	0,739	0,772	0,747	0,711
EnaA					
ω	popolna enakost	edit-distance	LCS	WLCS	skip-bigram
10	0,190	0,221	0,227	0,222	0,214
1	0,294	0,404	0,425	0,406	0,387
0,1	0,374	0,545	0,575	0,546	0,520
0,01	0,404	0,579	0,613	0,581	0,554
0,001	0,419	0,576	0,606	0,577	0,554
0,0001	0,418	0,558	0,585	0,559	0,539
0,00001	0,418	0,534	0,555	0,534	0,519

označuje povprečno podobnost za to metodo vrednotenja. Tabela 6.8 vsebuje samo povprečno podobnost za vseh 3000 razpletenih sej, grafi na slikah 6.9 in 6.10 pa prikazujejo razporeditev sej glede na stopnjo podobnosti.

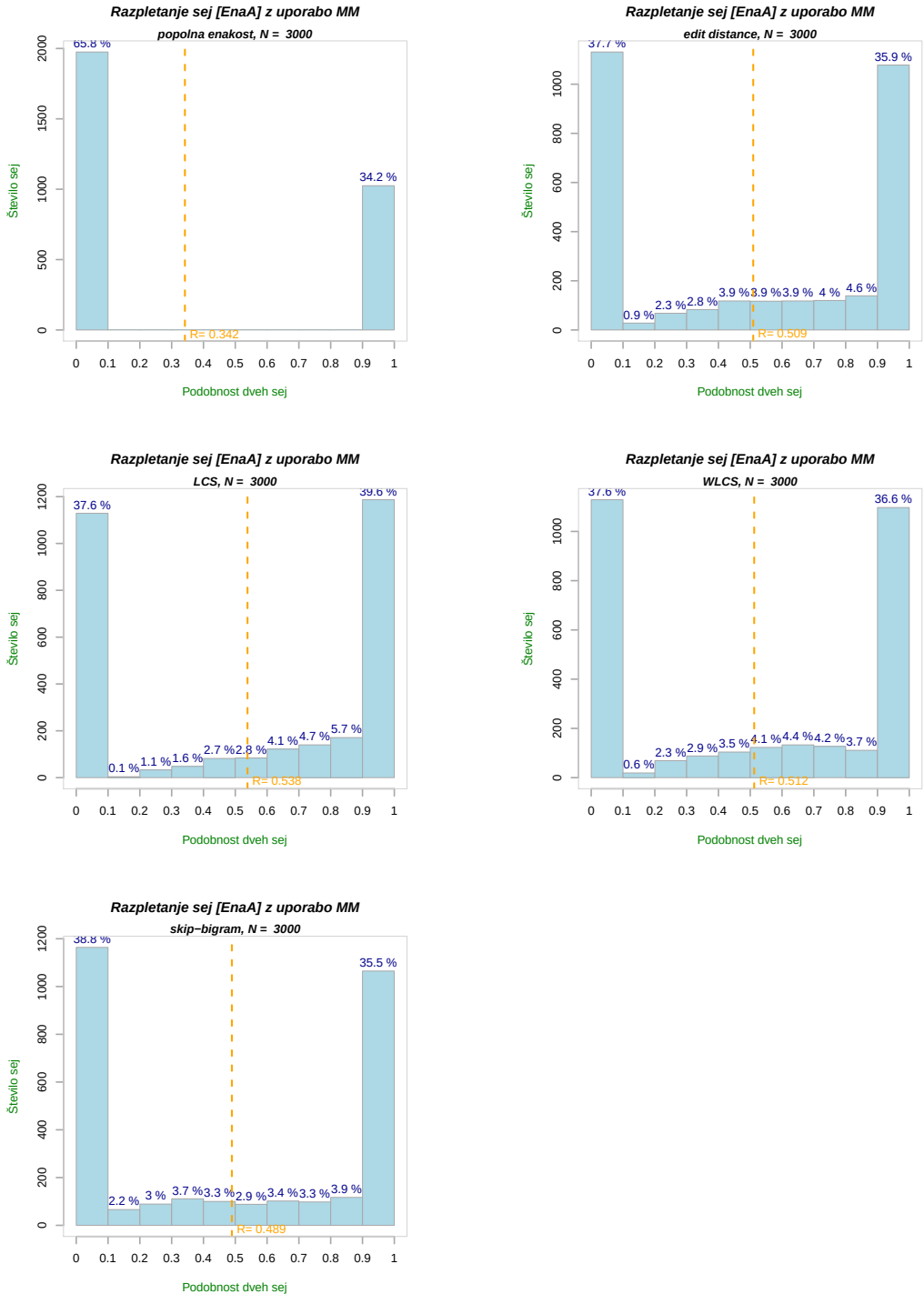
Pri razpletanju bi želeli čimbolj povečati število razpletenih sej s s podobnostjo blizu 1 in čimbolj zmanjšati število sej s podobnostjo blizu 0. Boljši rezultati razpletanja so tisti, kjer ima čimveč razpletov podobnost bližje 1. Idealni grafi na sliki 6.9 bi imeli en sam stolpec s podobnostjo razpletenih sej med 0,9 in 1. Če pogledamo istoležne grafe za oba vira podatkov vidimo, da so rezultati razpletanja sej sistema e-Študent boljši.

Tabela 6.8: Rezultati razpletanja prepletenih sej z uporabo markovskega modela. Vrednosti predstavljajo povprečno podobnost med razpletenimi in dejanskimi sestavnimi sejami.

vir	popolna enakost	edit-distance	LCS	WLCS	skip-bigram
e-Študent	0,461	0,704	0,726	0,692	0,689
EnaA	0,342	0,509	0,538	0,512	0,489



Slika 6.9: *Rezultati razpletanja prepletenih sej sistema e-Študent z uporabo MM. Uporabljene so različne mere podobnosti med razpleti: popolna enakost, razdalja med zaporedji, LCS, WLCS in skip-bigram.*



Slika 6.10: *Rezultati razpletanja prepletenih sej spletne trgovine z uporabo MM. Uporabljene so različne mere podobnosti med razpleti: popolna enakost, razdalja med zaporedji, LCS, WLCS in skip-bigram.*

Osredotočimo se na grafa za metodo LCS obeh virov podatkov. Najbolj nezaželjeni so razpleti s podobnostjo med 0 in 0,1. Pri sistemu e-Študent leži v tem intervalu podobnosti 23% sej, pri EnaA pa 37%, kar je skoraj dvakrat več. Na drugi strani imamo pri sistemu e-Študent veliko več sej v intervalu med 0,8 in 1. To pomeni, da je veliko sej sicer napačno razpletenih, vendar imajo zelo dolgo najdaljše skupno podzaporedje strani. Podobno lahko sklepamo na osnovi ostalih metod vrednotenja.

Vrednotenje glede na število sej v prepletu

Rezultate razpleta z uporabo MM, glede na število vsebovanih elementarnih sej v prepletu, prikazuje tabela 6.9. Uporabljeno je vrednotenje na podlagi popolne enakosti (100% pravilnost ali napačnost). Prva vrstica tabele 6.9 vsebuje podatke za sistem e-Študent, druga vrstica pa za spletno trgovino EnaA. V vsaki skupini je 1000 prepletenih sej. Pri obeh virih podatkov število pravilno razpletenih sej pada s številom vsebovanih čistih sej v prepletu, kar je pričakovano. Večje kot je število prepletenih sej, težje je pravilno določiti začetno stran vsake izmed njih. Pri razporejanju naslednje strani prepletene seje med tri razpletene seje je večja možnost napake kot v primeru dve razpletenih sej.

Najmanjša razlika napak pri prepletanju med obema viroma je pri neprepletenih sejah. Če se osredotočimo na prepletene seje z dvema prepletajočima se čistima sejama vidimo, da se pri sistemu e-Študent razplete popolnoma pravilno skoraj dvakrat več sej. Pri prepletih z večjim številom čistih sej je ta razlika še mnogo večja. Na podlagi teh rezultatov vidimo, da je klikotok spletne trgovine veliko bolj neugoden vir za razpletanje, saj je strani v sistemu veliko več, začetne strani pa je težje pravilno določiti.

Tabela 6.9: Rezultati postopka razpletanja z uporabo MM glede na število vsebovanih elementarnih sej.

vir podatkov	št. elementarnih sej		
	1	2	3
e-Študent	0,844	0,400	0,140
EnaA	0,792	0,208	0,027

6.5 Rezultati razpletanja pri uporabi RBFS

Prepletene seje, ki smo jih uporabili pri razpletanju v prejšnjem podpoglavju, smo razpletli še z uporabo metode RBFS. Množica prepletenih sej je bila ravno tako sestavljena iz treh skupin prepletov glede na število vsebovanih elementarnih sej. Rezultati razpleta za oba vira podatkov se nahajajo v tabeli 6.10. Prva vrstica tabele

vsebuje rezultate za sistem e-Študent, druga pa za spletno trgovino. Pomen stolpcev tabele 6.10 je enak kot pri tabeli 6.8, saj smo rezultate vrednotili na podoben način kot pri razpletanju z MM.

Tabela 6.10: Rezultati razpletanja prepletenih sej z uporabo postopka RBFS. Vrednosti predstavljajo povprečno podobnost med razpletenimi in dejanskimi sestavnimi sejami.

vir	popolna enakost	edit-distance	LCS	WLCS	skip-bigram
e-Študent	0,500	0,743	0,781	0,755	0,714
EnaA	0,405	0,565	0,596	0,567	0,543

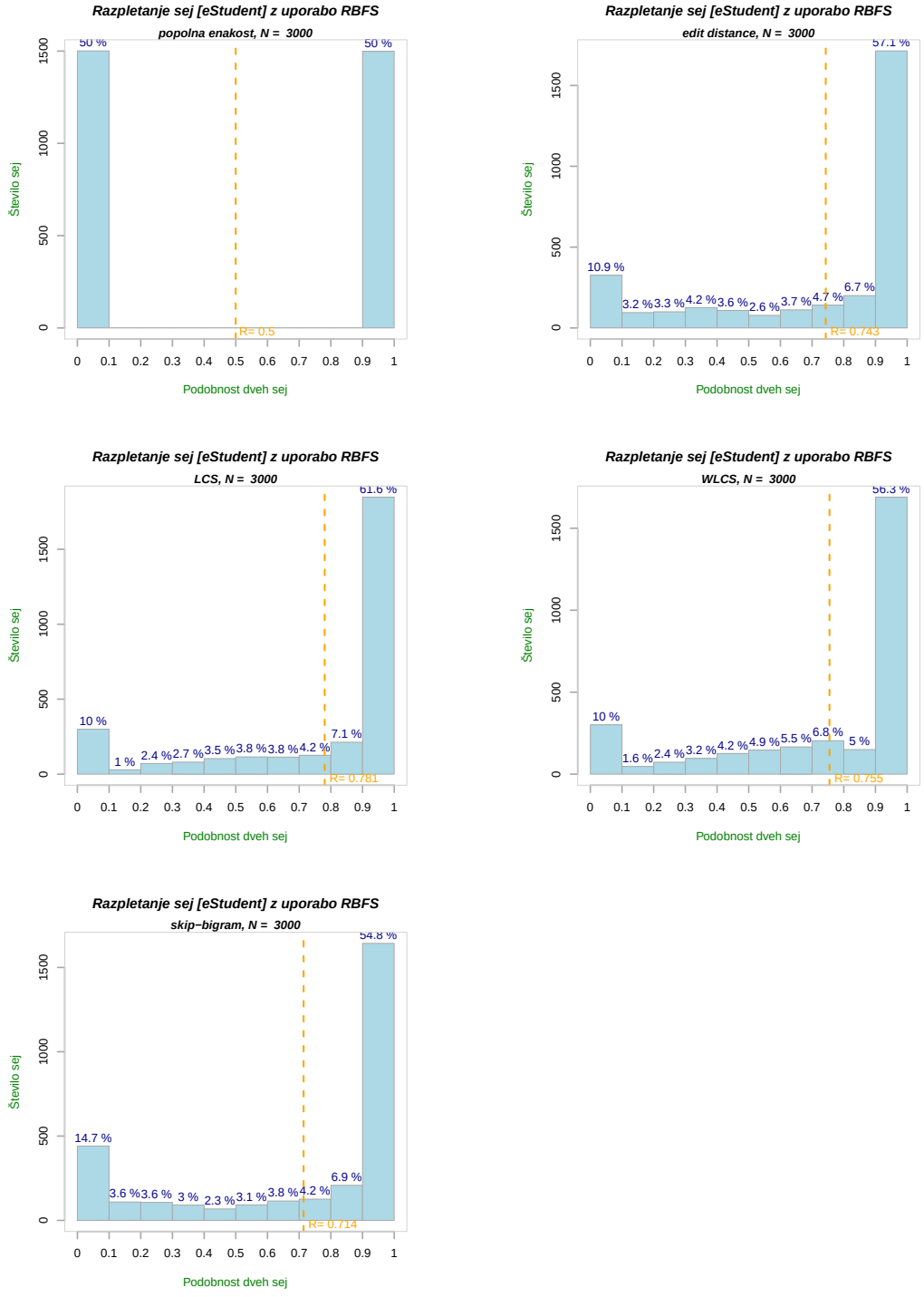
Tudi pri razpletanju z RBFS smo zabeležili boljše rezultate razpletanja za sistem e-Študent kot za spletno trgovino. Popolnoma pravilno smo razpletli polovico vseh sej sistema e-Študent, pri EnaA pa okrog 40% prepletenih sej. Podobno sliko kaže primerjava rezultatov ostalih metod vrednotenja za oba vira podatkov. Kot rečeno, je razlog večja strukturiranost sej sistema e-Študent.

Sliki 6.11 in 6.12 prikazujeta grafe metod vrednotenja rezultatov razpletanja prepletenih sej z uporabo RBFS pri vrednosti faktorja $\omega = 0,001$. Ovrednoteni rezultati za sistem e-Študent so prikazani na sliki 6.11, za spletno trgovino EnaA pa na sliki 6.12. Grafi in pomen oznak na obeh slikah so podobni kot pri grafih na slikah 6.9 in 6.10, ki prikazujejo rezultate razpletanja z metodo MM.

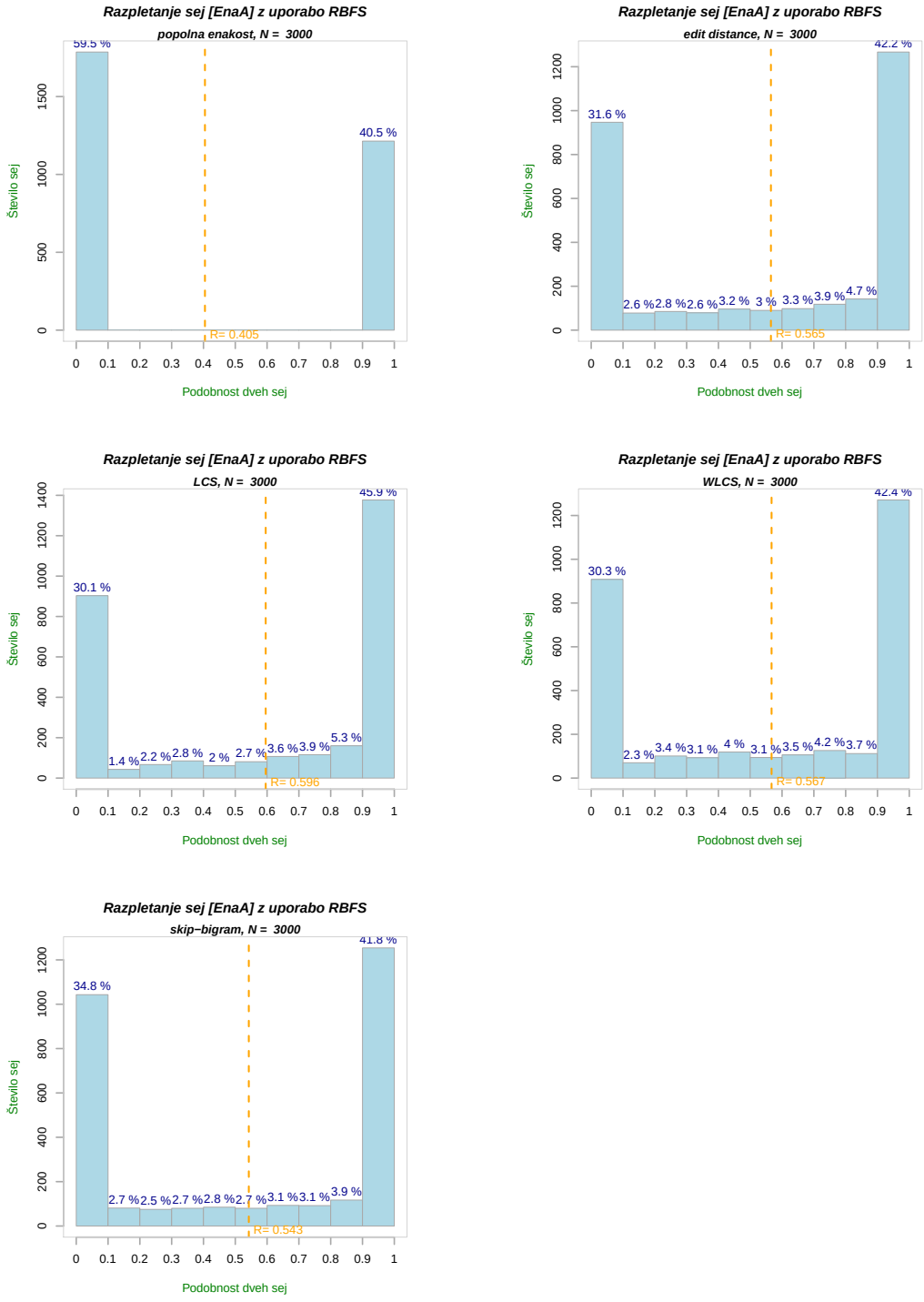
Primerjava grafov rezultatov razpleta z uporabo RBFS pokaže, da je kakovost razpletanja prepletenih sej pri sistemu e-Študent veliko večja kot je videti samo iz podatka o številu popolnoma pravilno razpletenih sej. Omejimo se na razpletene seje s podobnostjo med 0 in 0,1 ter metodo vrednotenja LCS. Porazdelitev sej s podobnostjo med 0,1 in 0,9 je podobna za oba vira podatkov. Najopaznejša razlika je pri sejah s podobnostjo med 0 in 0,1 ter pri sejah s podobnostjo med 0,9 in 1. Pri sistemu e-Študent je zelo slabo razpletenih sej s podobnostjo med 0 in 0,1 kar trikrat manj (10%) kot pri EnaA (30,1%). To so seje, ki imajo zelo malo skupnega s željenimi razpletenimi sejami, zato želimo njihovo število čimbolj zmanjšati. Pri sistemu e-Študent je za 15% večje tudi število zelo dobro razpletenih sej, s podobnostjo med 0.9 in 1. Do podobnih zaključkov lahko pridemo na osnovi ostalih metod vrednotenja.

Vrednotenje glede na število sej v prepletu

Rezultate razpletanja z uporabo RBFS glede na število vsebovanih elementarnih sej v prepletu prikazuje tabela 6.11. Pomen stolpcev je enak kot pri tabeli 6.9. Uporabljeno je vrednotenje na podlagi popolne enakosti. Prepletene seje so enakomerno razporejene med vse tri skupine. Imamo torej 1000 neprepletenih sej, 1000 sej iz dveh prepletajočih sej, in 1000 prepletov iz treh prepletajočih sej. Vidimo, da smo tudi pri uporabi RBFS najbolj uspešni pri identifikaciji neprepletenih sej. Slabše,



Slika 6.11: *Rezultati razpletanja prepletenih sej sistema e-Študent z uporabo RBFS. Uporabljene so različne mere podobnosti med razpleti: popolna enakost, razdalja med zaporedji, LCS, WLCS in skip-bigram.*



Slika 6.12: *Rezultati razpletanja prepletenih sej spletne trgovine z uporabo RBFS. Uporabljene so različne mere podobnosti med razpleti: popolna enakost, razdalja med zaporedji, LCS, WLCS in skip-bigram.*

vendar še vedno dobro, se razpletejo prepleti z dvema prepletajočima sejama, manj uspešni smo pri treh prepletajočih se sejah.

Kot smo videli v razdelku 3.4.3, število vseh možnih razpletov prepletene seje s številom razpletov hitro narašča, kar pomeni, da več kot je vsebovanih sej v prepletu, težje je prepleteno sejo pravilno razplesti v sestavne seje.

Tabela 6.11: Rezultati postopka razpletanja z uporabo RBFS glede na število vsebovanih elementarnih sej.

vir podatkov	št. elementarnih sej		
	1	2	3
e-Študent	0,941	0,451	0,112
EnaA	0,867	0,304	0,048

6.6 Diskusija

Obe metodi razpletanja sta se izkazali kot zelo uspešni, saj obe popolnoma pravilno razpleteta več kot 34% prepletenih sej spletne trgovine in 46% prepletenih sej sistema e-Študent. Obe metodi delujeta neodvisno od števila vsebovanih prepletenih sej v prepletih. To smo pokazali s testno množico prepletenih sej. Podatek o največjem številu prepletenih sej ni podan kot parameter nobene izmed obeh metod, kar še povečuje vrednost doseženih rezultatov. Obe metodi sta uporabni za poljuben vir klikotoka, boljše rezultate pa dosegata za uporabniške seje, ki imajo jasno določeno strukturo.

Primerjava rezultatov razpletanja med metodama MM in RBFS pokaže, da metoda RBFS daje boljše rezultate za oba vira podatkov glede na vseh 5 kriterijev ocenjevanja podobnosti med razpletenimi in dejanskimi sestavnimi sejami.

Z uporabo metode RBFS smo popolnoma pravilno razpletli 50% sej prepletov sistema e-Študent, z uporabo metode MM pa 46%. Razlika v kakovosti razpletanja postane še bolj očitna, če primerjamo grafe rezultatov na slikah 6.9 in 6.11. Omejimo se na zelo slabo razpletene seje s podobnostjo od 0 do 0,1 in metodo vrednotenja LCS. Pri uporabi MM pade v ta interval 23% veliko sej, pri uporabi RBFS pa več kot dvakrat manj (10%). Z RBFS torej dosežemo večjo podobnost razpletenih sej dejanskim razpletom za seje, ki niso bile popolnoma pravilno razpletene.

Podobno velja za spletno trgovino EnaA. Razpletanje z uporabo RBFS vrne kot rezultat 40% popolnoma pravilno razpletenih sej, razpletanje z uporabo MM pa 34%. Razlika v številu zelo slabo razpletenih sej (podobnost med 0 in 0,1) je manjša kot pri sistemu e-Študent, vendar še vedno 7%. Pri uporabi RBFS pade v ta interval podobnosti 30,1% sej, pri uporabi MM pa 37,6. V interval podobnosti med 0,9 in 1 pade pri uporabi RBFS 45,9% sej, pri uporabi MM pa 39,6%. Število sej v intervalu podobnosti med 0,1 in 0,9 je primerljivo pri obeh metodah.

Postopek RBFS deluje po načelu najbolj verjetenega razpleta. To pomeni, da išče razpletene seje z največjim produktom verjetnosti sej, medtem ko metoda z uporabo markovskega modela deluje lokalno na podlagi verjetnosti prehoda med dvemi stranmi.

Postopek razpletanja z uporabo MM je prostorsko in časovno učinkovit. Za razpletanje in ovrednotenje 3000 razpletenih sej porabi manj kot sekundo časa na povprečnem računalniku, medtem ko je postopek razpletanja sej z RBFS časovno precej bolj zahteven. Za razpletanje in ovrednotenje 3000 sej smo potrebovali v povprečju 4 ure in pol.

POGLAVJE 7

Zaključek

V doktorskem delu smo se ukvarjali z razpletenjam prepletenih HTTP sej, ki nastanejo pri istočasnem deskanju uporabnika z več odprtimi okni brskalnika po določenem spletnem mestu. Prepletene seje kvarno vplivajo na kakovost analiz podatkov klikotoka. Cilj doktorata je bil razviti metode za izboljšanje kakovosti podatkov v postopku predprocesiranja podatkov o klikotoku. Preučili smo možnosti za razplet prepletenih sej, razvili algoritme za razpletanje in eksperimentalno potrdili metode na realnih podatkih o klikotoku. V zadnjem poglavju disertacije bomo navedli rezultate, povzeli glavne prispevke k znanosti in podali predloge za nadaljne delo na področju razpletanja prepletenih spletnih sej.

7.1 Rezultati

Pri analizi podatkov o klikotoku smo naleteli na problem prepletenih HTTP sej, ki jih ni mogoče preprosto ločiti v postopku osejevanja. V tem delu smo se ukvarjali z razpletanjem takšnih sej. Preučili smo obnašanje uporabnikov, ki generirajo prepletene seje, analizirali prepletene seje in ugotovili ključne probleme pri njihovem razpletanju. Problem razpletanja smo osvetlili s teoretičnega vidika. Definirali smo formule za izračun števila vseh možnih prepletov dveh sej in vseh različnih razpletov prepletene seje ter dokazali povezavo z Bellovimi in Strilingovimi števili drugega reda. S kombinatoričnimi izračuni smo pokazali zapletenost problema razpletanja sej.

V literaturi smo preučili, kateri pristopi se uporabljajo pri reševanju problemov na sorodnih področjih. Preverjene pristope smo nadgradili z izvirnimi rešitvami in razvili dve metodi za razpletanje prepletenih sej. Prva metoda temelji na markovskem modelu in ima linearno časovno zahtevnost. Pri drugi metodi smo problem razpletanja sej prevedli na problem preiskovanja prostora stanj. Razvili smo po-

polno hevristiko za reševanje problema razpletanja sej, ki učinkovito usmerja iskanje v prostoru stanj. Postopek razpletanja razplete popolnoma pravilno 50% prepletenih sej, ki imajo vsebovano različno število elementarnih sej. Brez postopka razpletanja bi se te seje v podatkovno skladišče zapisale kot neprepletene seje. Rezultat je toliko boljši, če upoštevamo, da se popolnoma pravilno razplete 45% prepletenih sej z dvema prepletenima sejama, 11 % sej s tremi prepletenimi sejami, pri tem pa 94 % neprepletenih sej ostane nespremenjenih. V praksi ne potrebujemo vedno samo popolnoma pravilno razpletenih sej. Velikokrat je dovolj, če je pravilno razpleten samo velik del posamezne uporabniške seje. Vrednotenje rezultatov na podlagi podobnosti sej nam da še boljše rezultate. Razpletene in dejanske seje imajo na podlagi vrednotenja z metodo LCS v povprečju kar 78% podobnost. Predstavili smo tudi postopke za ovrednotenje kakovosti razpletanja.

Razviti metodi smo preverili na umetnih in realnih podatkih o klikotoku. Umetne podatke o klikotoku smo generirali na podlagi analize realnih podatkov in lastnosti, ki veljajo za realne čiste in prepletene seje. Za realni vir podatkov smo izbrali klikotok dveh spletnih aplikacij, ki generirata po svoji naravi zelo različen klikotok: spletni študijski informacijski sistem in spletno trgovino. Pokazali smo, da metodi dobro delujeta na različnih vrstah podatkov o klikotoku in jih lahko uporabimo na poljubnem viru podatkov o klikotoku.

S pomočjo razvitih metod razpletanja smo izboljšali postopek predprocesiranja podatkov o klikotoku in rešili problem prepletenih sej v spletnem podatkovnem skladišču. Spletno podatkovno skladišče zato vsebuje kakovostnejše podatke ter omogoča kakovostnejše analize.

7.2 Prispevki k znanosti

Na podlagi rezultatov v prejšnjem razdelku navedimo strnjen pregled prispevkov doktorske disertacije:

- Pregled področij teme. Naredili smo celovit pregled del avtorjev, ki se ukvarjajo s področjem klikotoka ter s klikotokom povezanimi področji. Osvetlili smo glavne probleme, s katerimi se raziskovalci ukvarjajo, in do sedaj uporabljene pristope. Sistematičen pregled področij predstavlja temelj za nadaljnje delo na področju klikotoka.

- Analiza težav pri zajemu in pripravi podatkov iz spletnih dnevnikov. Predstavili smo klikotok ter glavne težave pri čiščenju in preoblikovanju podatkov. Pokazali smo, da je enolična identifikacija uporabniških sej v praksi nemogoča brez neke vrste kontekstne pomoči.

- Razvoj algoritmov za razpletanje prepletenih uporabniških sej. Na podlagi poznavanja podatkov o klikotoku in obnašanja uporabnikov smo razvili dve različni metodi za razpletanje prepletenih sej. Obe metodi sta se pokazali kot

uspešni pri izboljšavi kakovosti podatkov v spletnem podatkovnem skladišču. Njuna prednost je enostavna umestitev v postopek procesiranja podatkov o klikotoku in predvsem popolnoma pravilno razpletanje velikega števila prepletenih sej.

- Razvoj popolne hevristične funkcije za hevristično iskanje. Problem razpletanja sej smo prevedli na problem preiskovanja prostora stanj. Razvili smo popolno hevristiko, ki omogoča učinkovito iskanje ciljnih razpletov.

- Razvoj realističnega modela za vrednotenje delovanja zgornjega algoritma. Za prikaz, da razviti metodi razpletanja sej kakovostno razpletata prepletene seje, smo razvili model za vrednotenje razpletanja po različnih kriterijih za merjenje podobnosti med razpletenimi in originalnimi sejami. Z njimi smo pokazali realno kakovost postopkov razpletanja.

- Empirični preizkus izdelanih postopkov razpletanja v praksi. Razvite postopke smo preverili na realnih primerih prepletenih sej spletne trgovine in spletnega študijskega IS. Pokazali smo, da sta metodi uporabni na poljubnem viru podatkov o klikotoku.

7.3 Možnosti za nadaljnje delo

Motivacija za nadaljnje delo je povezana predvsem z izboljšanjem točnosti razpletanja prepletenih sej.

- Izboljšava načina identifikacije začetnih stanj. Največja težava pri razpletanju sej je pravilna identifikacija začetnih stanj razpletenih sej. Nepravilno identificirano začetno stanje pomeni tudi napačen celoten razplet. Zasnovati bi bilo potrebno postopek za bolj natančno določanje začetnega stanja.

- Odprava problema časovne prekinitve pri razpletanju s pomočjo preiskovanja prostora stanj. Postopek preiskovanja bi nadgradili z uporabo omejenega iskanja v globino, ki se običajno uporablja pri igrah z dvema igralcema, kot je recimo šah. Pri tem bi lahko uporabili izpeljanko algoritma Real-Time-A* (RTA*).

- Uporaba vira zahteve (ang. referrer) pri razpletanju. Pri dnevnikih spletnih mest, kjer je na voljo podatek o viru zahteve, bi ta podatek lahko uporabili pri izboljšanju natančnosti razpletanja.

Na ta način bi nadgradili in izboljšali rezultate disertacije. Metode bi lahko preiskusili tudi na drugih realnih virih podatkov in potrdili ustreznost pristopa.

Literatura

[1] S. Abiteboul, I Manolescu, in N. Preda. Constructing and Querying Peer-to-Peer Warehouses of XML Resources. *Data Engineering, International Conference on*, 0:1122–1123, 2005.

[2] M. Abramowitz in I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications, New York, ninth dover printing, tenth gpo printing izdaja, 1972.

[3] P. Adamlje. Prototip agenta za zasledovanje obiskovalcev spleta in uporaba sledi v podatkovnem skladišču, 2007.

[4] D. Agrawal. The Reality of Real-time Business Intelligence. *BIRTE (Informal Proceedings)*, 2008.

[5] C.R. Anderson, P. Domingos, in D.S. Weld. Relational Markov models and their application to adaptive web navigation. V *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, stran 152. ACM, 2002.

[6] D.P. Ballou in G.K. Tayi. Enhancing data quality in data warehouse environments. *Commun. ACM*, 42(1):73–78, 1999.

[7] A. Banerjee in J. Ghosh. Clickstream clustering using weighted longest common subsequences. V *Proc. of the Workshop on Web Mining, SIAM Conference on Data Mining*, strani 33–40. Citeseer, 2001.

[8] B. Berendt, B. Mobasher, M. Nakagawa, in M. Spiliopoulou. The impact of site structure and user environment on session reconstruction in web usage analysis. V *WEBKDD - KDD Workshop on Web Mining and Web Usage Analysis*, strani 159–179, 2002.

[9] B. Berendt, B. Mobasher, M. Spiliopoulou, in J. Wiltshire. Measuring the accuracy of sessionizers for web usage analysis. V *Workshop on Web Mining at the First SIAM International Conference on Data Mining*, strani 7–14, 2001.

[10] I. Bratko. *Prolog Programming for Artificial Intelligence*. Pearson Addison-Wesley, Harlow, England, 3. izdaja, 2000.

[11] R.M. Bruckner, B. List, in J. Schiefer. Striving towards near real-time data integration for data warehouses. V *DaWaK 2000: Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery*, strani 317–326, London, UK, 2002. Springer-Verlag.

[12] I. Cadez, D. Heckerman, C. Meek, P. Smyth, in S. White. Visualization of navigation patterns on a web site using model-based clustering. V *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, strani 280–284. ACM New York, NY, USA, 2000.

[13] L.D. Catledge in J.E. Pitkow. Characterizing browsing strategies in the world-wide web. *Computer Networks and ISDN Systems*, 27(6):1065–1073, April 1995.

[14] P. Chapman, J. Clinton, T. Khabaza, Reinartz T., in R. Wirth. *The CRISP-DM Process Model*. CRISP-DM consortium, marec 1999.

[15] E. Charniak, S. Goldwater, in M. Johnson. Edge-based best-first chart parsing. V *Proceedings of the Sixth Workshop on Very Large Corpora*, strani 127–133, 1998.

[16] S. Chaudhuri in U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Rec.*, 26(1):65–74, 1997.

[17] S. Chaudhuri, U. Dayal, in V. Ganti. Database technology for decision support systems. *Computer*, 34(12):48–55, Dec 2001.

[18] K. Chenoweth, T.and Corral in H. Demirkan. Seven key interventions for data warehouse success. *Commun. ACM*, 49(1):114–119, 2006.

[19] J. H. Conway in R. K. Guy. *The Book of Numbers*. Springer-Verlag, New York, 1995.

[20] R. Cooley, B. Mobasher, in J. Srivastava. Data Preparation for Mining World Wide Web Browsing Patterns. *Knowledge and Information Systems*, 1:5–32, 1999.

[21] T. H. Cormen, C. E. Leiserson, in R. L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1989.

[22] F. J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, 1964.

[23] M. Deshpande in G. Karypis. Selective Markov models for predicting Web page accesses. *ACM Transactions on Internet Technology (TOIT)*, 4(2):163–184, 2004.

[24] L. Di Scala, L. La Rocca, in G. Consonni. A Bayesian hierarchical model for the evaluation of a website. *Journal of Applied Statistics*, 31(1):15–28, 2004.

[25] T.G. Dietterich. Machine learning for sequential data: A review. *Lecture Notes in Computer Science*, strani 15–30, 2002.

[26] Gambit Trade d.o.o. Spletna trgovina EnaA. `http://www.enaa.com/`, apr 2010.

[27] G. Druck, MA Amherst, M. Narasimhan, WA Redmond, in P. Viola. Learning A* underestimates: Using inference to guide inference. V *Proc. AI-STATS*. Citeseer, 2007.

[28] I.S. Duff, AM Erisman, in J.K. Reid. *Direct methods for sparse matrices*. Oxford University Press, USA, 1989.

[29] S. Dzeroski, B. Cestnik, in I. Petrovski. Using the m-estimate in rule induction. *Journal of Computing and Information Technology*, 1(1):37–46, 1993.

[30] W.W. Eckerson. *Evolution of data warehousing: the trend toward analytical applications*, stran 1–8. The Patricia Seybold Group, April 1999.

[31] M. Eirinaki, M. Vazirgiannis, in D. Kapogiannis. Web path recommendations based on page ranking and Markov models. V *Proceedings of the 7th annual ACM international workshop on Web information and data management*, stran 9. ACM, 2005.

[32] L.P. English. *Improving data warehouse and business information quality: methods for reducing costs and increasing profits*. John Wiley & Sons, Inc. New York, NY, USA, 1999.

[33] K.D. Fenstermacher in M. Ginsburg. Client-side monitoring for web mining. *Journal of the American Society for Information Science and Technology*, 54(7):625–637, 2003.

[34] P. Fraternali. Tools and approaches for developing data-intensive web applications: a survey. *ACM Comput. Surv.*, 31(3):227–263, 1999.

[35] G. Furlow. The case for building a data warehouse. *IT PROFESSIONAL*, strani 31–34, 2001.

[36] R. L. Graham, D. E. Knuth, in O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[37] G. Gundersen in T. Steihaug. Data structures in Java for matrix computations. *Concurrency and Computation Practice and Experience*, 16(8):799–815, 2004.

[38] Ş. Gündüz in M.T. Özsu. A web page prediction model based on click-stream tree representation of user behavior. V *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, stran 540. ACM, 2003.

[39] W. H. Inmon. *Building the Data Warehouse,3rd Edition*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[40] I.C. Italiano in J.E. Ferreira. Synchronization options for data warehouse designs. *Computer*, 39(3):53–57, March 2006.

[41] A. Karakasidis, P. Vassiliadis, in E. Pitoura. ETL queues for Active Data Warehousing. V *IQIS '05: Proceedings of the 2nd international workshop on Information quality in information systems*, strani 28–39, New York, NY, USA, 2005. ACM.

[42] N. Khasawneh in C.C. Chan. Active user-based and ontology-based web log data preprocessing for web usage mining. V *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, strani 325–328. IEEE Computer Society, 2006.

[43] R. Kimball in R. Merx. *The Data Webhouse Toolkit – Building Web-Enabled Data Warehouse*. Wiley Computer Publishing, New York, August 2002 2000.

[44] R. Kimball, L. Reeves, W. Thornthwaite, M. Ross, in W. Thornwaite. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[45] R. Kimball in M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[46] D. Klein in C.D. Manning. Accurate unlexicalized parsing. V *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, strani 423–430. Association for Computational Linguistics, 2003.

[47] R. Kohavi. Mining e-commerce data: The good, the bad, and the ugly. V Foster Provost in Ramakrishnan Srikant, uredniki, *Proceedings of the Seventh*

ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, strani 8–13, 2001.

[48] G. Leusch, N. Ueffing, in H. Ney. A novel string-to-string distance measure with applications to machine translation evaluation. V *In Proceedings of MT Summit IX*, strani 240–247, 2003.

[49] M. Levene in G. Loizou. Computing the entropy of user navigation in the web. *International Journal of Information Technology and Decision Making*, 2(3):459–476, 2003.

[50] M. Levene in G. Loizou. Why is the snowflake schema a good data warehouse design? *Inf. Syst.*, 28(3):225–240, 2003.

[51] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, 1966.

[52] C-Y. Lin in F. J. Och. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. V *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, stran 605, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

[53] X. Liu, J. Heo, in L. Sha. Modeling 3-Tiered Web Applications. *mascots*, 0:307–310, 2005.

[54] C.T. Lopes in G. David. *Higher Education Web Information System Usage Analysis with a Data Webhouse*, zvezek 3983 of *Lecture Notes in Computer Science*, strani 78–87. Springer, Berlin / Heidelberg, May 2006.

[55] Oracle Ltd. A Data Mart Concepts. `http://download.oracle.com/docs/cd/E10352_01/doc/bi.1013/e10312/dm_concepts.htm`, april 2010.

[56] L. Lu, M. Dunham, in Y. Meng. Discovery of significant usage patterns from clusters of clickstream data. V *Workshop Notes*, stran 68. Citeseer, 2005.

[57] H.P. Luhn. A business intelligence system. *IBM Journal of Research and Development*, 2(4):314–319, 1958.

[58] M. Lujan, A. Usman, P. Hardie, TL Freeman, in J. Gurd. Storage formats for sparse matrices in Java. *Computational Science–ICCS 2005*, strani 364–371, 2005.

[59] A. Luotonen. The Common Log File format. *CERN httpd user manual*, 1995.

[60] V. Mahnič in I. Rožanc. Data Quality: A Prerequisite for Successful Data Warehouse Implementation. *Informatica, Special issue: The Changing University, and the Role of Information Technology, zv*, 25:183–188.

[61] V. Mahnič, I. Rožanc, in M. Poženel. Using e-business technology in a student records information system. V *Recent advances in e-activities : proceedings of the 7th WSEAS International Conference on E-Activities (E-Activities'08)*, strani 80–85, Cairo, Egypt, December 29–31 2008.

[62] S.T. March in A.R. Hevner. Integrated decision support systems: A data warehousing perspective. *Decision Support Systems*, 43(3):1031–1043, 2007.

[63] B.M. Masand, M. Spiliopoulou, J. Srivastava, in O.R. Zaiane. Webkdd 2002: Web mining for usage patterns & profiles. *SIGKDD Explor. Newsl.*, 4(2):125–127, 2002.

[64] A.L. Montgomery, S. Li, K. Srinivasan, in J.C. Liechty. Modeling online browsing and path analysis using clickstream data. *Marketing Science*, 23(4):579–595, 2004.

[65] S. Negash in P. Gray. Business intelligence. *Communications of the Association for Information Systems*, 13(1):15, 2004.

[66] Ltd. Netcraft. Web Server Survey. `http://news.netcraft.com/`, jan 2010.

[67] K. Papineni, S. Roukos, T. Ward, in W-J. Zhu. Bleu: a method for automatic evaluation of machine translation. V *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, strani 311–318, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[68] L.D. Paulson. Building rich web applications with Ajax. *Computer*, 38(10):14–17, 2005.

[69] T.B. Pedersen in C.S. Jensen. Multidimensional database technology. *Computer*, 34(12):40–46, Dec 2001.

[70] M. Perkowitz in O. Etzioni. Adaptive web sites: an AI challenge. V *IJCAI-97: proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, Nagoya, Japan, August 23-29, 1997*, stran 16. IJCAII, 1997.

[71] D. Pierrakos, G. Paliouras, C. Papatheodorou, in C.D. Spyropoulos. Web usage mining as a tool for personalization: A survey. *User Modeling and User-Adapted Interaction*, 13(4):311–327, 2004.

[72] J. Pitkow in P. Pirolli. Mining longest repeating subsequences to predict World Wide Web surfing. V *USITS'99: Proceedings of the 2nd conference*

on USENIX Symposium on Internet Technologies and Systems, strani 13–13, Berkeley, CA, USA, 1999. USENIX Association.

[73] M. Poess in R.O. Nambiar. Large scale data warehouses on grid: Oracle database 10g and hp proliant servers. V *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, strani 1055–1066. VLDB Endowment, 2005.

[74] N. Polyzotis, S. Skiadopoulos, P. Vassiliadis, A. Simitsis, in N.E. Frantzell. Supporting Streaming Updates in an Active Data Warehouse. V *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, strani 476–485, April 2007.

[75] M. Poženel in V. Mahnič. Data webhouse: A tool for monitoring the use of a web based information system. V *Proceedings of the 13th International Conference EUNIS 2007: IT innovation for a European era*, strani 100–112, Grenoble, june 2007.

[76] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Readings in speech recognition*, 53(3):267–296, 1990.

[77] J. Riordan. *Combinatiorial Identities*. Wiley, New York, USA, 1979.

[78] S. Rizzi, A. Abelló, J. Lechtenbörger, in J. Trujillo. Research in data warehouse modeling and design: dead or alive? V *DOLAP '06: Proceedings of the 9th ACM international workshop on Data warehousing and OLAP*, strani 3–10, New York, NY, USA, 2006. ACM.

[79] S. Roman. *The Umbral Calculus*. Dover Publications, 2005.

[80] G-C. Rota. The number of partitions of a set. *American Mathematical Monthly*, 71(5):498–504, 1964.

[81] S.J. Russell in P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[82] R. R. Sarukkai. Link prediction and path analysis using markov chains. V *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*, strani 377–386, Amsterdam, The Netherlands, The Netherlands, 2000. North-Holland Publishing Co.

[83] S. Schechter, M. Krishnan, in M.D. Smith. Using path profiles to predict HTTP requests. *Computer Networks and ISDN Systems*, 30(1-7):457–467, 1998.

[84] R. Sen in M.H. Hansen. Predicting Web Users' Next Access Based on Log Data. *Journal of Computational and Graphical Statistics*, 12(1):143–155, 2003.

[85] C. Shahabi, F. Banaei-Kashani, in J. Faruque. A framework for efficient and anonymous web usage mining based on client-side tracking. *Lecture Notes in Computer Science*, 2356:113–144, 2002.

[86] C. Shahabi, A.M. Zarkesh, J. Adibi, in V. Shah. Knowledge discovery from users web-page navigation. *Research Issues in Data Engineering, International Workshop on*, 0:20, 1997.

[87] A. Simitsis, P. Vassiliadis, in T. Sellis. Optimizing ETL processes in data warehouses. V *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, strani 564–575, April 2005.

[88] M. Spiliopoulou, B. Mobasher, B. Berendt, in M. Nakagawa. A framework for the evaluation of session reconstruction heuristics in web-usage analysis. *INFORMS Journal on Computing*, 15(2):171–190, 2003.

[89] J. Srivastava, R Cooley, M. Deshpande, in P.N. Tan. Web usage mining: discovery and applications of usage patterns from Web data. *SIGKDD Explor. Newsl.*, 1(2):12–23, 2000.

[90] D. Tanasa in B. Trousse. Advanced data preprocessing for intersites web usage mining. *IEEE Intelligent Systems*, 19(2):59–65, 2004.

[91] I.H. Ting, C. Kimble, in D. Kudenko. A pattern restore method for restoring missing patterns in server side clickstream data. *Lecture Notes in Computer Science*, 3399:501–512, March 2005.

[92] J. Trujillo in M. Palomar. An object oriented approach to multidimensional database conceptual modeling (OOMD). V *DOLAP '98: Proceedings of the 1st ACM international workshop on Data warehousing and OLAP*, strani 16–21, New York, NY, USA, 1998. ACM.

[93] J. P. Turian, L. S., in I. D. Melamed. Evaluation of machine translation and its evaluation. V *In MT Summit IX*, strani 386–393, 2003.

[94] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[95] P. Vassiliadis in T. Sellis. A survey of logical models for OLAP databases. *SIGMOD Rec.*, 28(4):64–69, 1999.

[96] S. Viaene. Linking business intelligence into your business. *IT Professional*, 10(6):28–34, Nov.-Dec. 2008.

[97] M. Viermetz, C. Stolz, V. Gedov, in M. Skubacz. Relevance and impact of tabbed browsing behavior on web usage mining. V *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, strani 262–269, Dec. 2006.

[98] P. Viola in M. Narasimhan. Learning to extract information from semi-structured text using a discriminative context free grammar. V *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, strani 330–337, New York, NY, USA, 2005. ACM.

[99] B. Vrdoljak, M. Banek, in S. Rizzi. Designing web warehouses from XML schemas. V *In Proc. DaWaK*, strani 89–98, 2003.

[100] H.J. Watson. Recent developments in data warehousing. *Communications of the Association for Information Systems*, 8(1):1–25, 2002.

[101] H.J. Watson in B.H. Wixom. The current state of business intelligence. *Computer*, 40(9):96–99, 2007.

[102] H.J. Watson, B.H. Wixom, in D.L. Goodhue. Data Warehousing: The 3M Experience. *Organizational Data Mining: Leveraging Enterprise Data Resources for Optimal Performance*, strani 202–216, 2004.

[103] R. Wirth in J. Hipp. CRISP-DM: Towards a standard process modell for data mining. V *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*, strani 29–39, april 2000.

[104] B.H. Wixom in H.J. Watson. An empirical investigation of the factors affecting data warehousing success. *MIS quarterly*, 25(1):17–41, 2001.

[105] T.W. Yan, M. Jacobsen, H. Garcia-Molina, in U. Dayal. From user access patterns to dynamic hypertext linking. *Computer Networks and ISDN Systems*, 28(7-11):1007–1014, 1996.

[106] A. Ypma in T. Heskes. Automatic categorization of web pages and user clustering with mixtures of hidden Markov models. *Lecture Notes in Computer Science*, 2703:35–49, 2003.

[107] J. Zhang in A.A. Ghorbani. The reconstruction of user sessions from a server log using improved time-oriented heuristics. V *Communication Networks and Services Research, 2004. Proceedings. Second Annual Conference on*, strani 315–322, May 2004.

[108] J. Zhu, J. Hong, in J. Hughes. Using Markov Chains for Link Prediction in Adaptive Web Sites. V *Soft-Ware*, strani 60–73. Springer, 2002.

[109] Y. Zhu, L. An, in S. Liu. Data Updating and Query in Real-Time Data Warehouse System. *Computer Science and Software Engineering, International Conference on*, 5:1295–1297, 2008.

Stvarno kazalo

Kratice, okrajšave, simboli

Kratica	Opis
BI	Business Intelligence
BPM	Business Process Monitoring
CIO	Chief Information Officer
CLF	Common Log Format
CRM	Customer Relationship Management
CSF	Critical Success Factors
DW	Data Warehouse
ECLF	Extended Common Log Format
EIS	Executive Information Systems
EM	Expectation-Maximization
ERP	Enterprise Resource Planning
ETL	Extraction, Transformation, Loading
HMM	Hidden Markov Model
HTTP	HyperText Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDA*	Iterative-deepening A*
IS	Information System
IT	Information Technology
JSA	Java Sparse Array
KPI	Key Performance Indicators
LCS	Longest Common Subsequence
MDB	Multidimensional Database
MM	Markov Model
ODS	Operational Data Store
OLAP	Online Analytical Processing

Kratica	Opis
OLTP	Online Transaction Processing
P2P	Peer-to-peer
PB	podatkovna baza
PS	podatkovno skladišče
RBFS	Recursive Best-first Search
RMM	Relational Markov Model
SOA	Service Oriented Architecture
SPS	spletno podatkovno skladišče
SSAS	SQL Server Analysis Services
SSIS	SQL Server Integration Services
SSNF	Snowflake Schema Normal Form
SQL	Structured Query Language
SUPB	sistem za upravljanje s podatkovno bazo
TPS	tipična prepletena seja
URL	Uniform Resource Locators
URI	Uniform Resource Identifier
WLCS	Weighted Longest Common Subsequence